

Rancher Red Book

[Draft]

Table of Contents

前言 1.1
Rancher 概述 1.2
快速安装指南 1.3
Rancher 安装 1.4
单节点服务器 1.4.1
多节点高可用 1.4.2
基本 SSL 配置 1.4.3
无互联网访问 1.4.4
添加主机 1.4.5
基本概念 1.5
Rancher 基础服务 1.6
健康检查 1.6.1
内部 DNS 服务 1.6.2
使用 Route 53 做外部 DNS 1.6.3
元数据服务 1.6.4
存储服务 1.6.5
审计日志 1.6.6
系统配置 1.7
访问控制 1.7.1
设置 1.7.2
账户 1.7.3
环境 1.7.4
API & Keys 1.7.5
镜像仓库 1.7.6
使用 Docker 原生命令行 1.8
使用 Rancher-Compose 1.9
使用 Rancher Compose 1.9.1
命令和选项 1.9.2
Rancher 服务 1.9.3
调度 1.9.4

- 升级服务 1.9.5
- 环境插值 1.9.6
- 构建使用 AWS S3存储 1.9.7
- Rancher 图形界面 1.10
  - 调度服务 1.10.1
  - 服务/堆栈 1.10.2
    - 增加服务 1.10.2.1
    - 增加服务别名 1.10.2.2
    - 增加负载均衡 1.10.2.3
    - 增加外部服务 1.10.2.4
    - 堆栈选项 1.10.2.5
  - 基础架构/主机 1.10.3
    - 从主机开始 1.10.3.1
    - 添加自定义主机 1.10.3.2
    - Amazon EC2 1.10.3.3
    - Azure 1.10.3.4
    - Digital Ocean 1.10.3.5
    - Exoscale 1.10.3.6
    - Packet 1.10.3.7
    - Rackspace 1.10.3.8
    - 其它驱动 1.10.3.9
  - 基础架构/容器 1.10.4
  - 基础架构/证书 1.10.5
- Rancher 目录 1.11
- Kubernets 1.12
- Swarm 1.13
- 使用标签 1.14
- 升级 Rancher 1.15
- 为 Rancher 做贡献 1.16
- 使用 API 1.17
  - 常用资源字段 1.17.1
  - API 资源 1.17.2
- Rancher OS 1.18
  - 快速安装指南 1.18.1

运行 Rancher OS 1.18.2
工作站 1.18.2.1
Docker Machine 1.18.2.1.1
Vagrant 1.18.2.1.2
从 ISO 启动 1.18.2.1.3
公有云 1.18.2.2
AWS 1.18.2.2.1
GCE 1.18.2.2.2
Azure 1.18.2.2.3
裸金属机和虚拟机 1.18.2.3
iPXE 1.18.2.3.1
PXE 1.18.2.3.2
安装到硬盘 1.18.2.3.3
配置 1.18.3
配置存储 1.18.3.1
网络 1.18.3.2
用户 1.18.3.3
SSH Keys 1.18.3.4
控制台输出 1.18.3.5
系统服务 1.18.3.6
设置 Docker TLS 1.18.3.7
加载内核模块 1.18.3.8
安装内核模块 1.18.3.9
DKMS 1.18.3.10
自定义内核 1.18.3.11
构建自定义 RancherOS ISO 1.18.3.12
Docker 1.18.3.13
定制 Docker 1.18.3.14
预打包 Docker 镜像 1.18.3.15
公有云配置参考 1.18.4
升级 1.18.5
ROS 工具 1.18.6
ROS 简介 1.18.6.1

Config 1.18.6.2
TLS 1.18.6.3
OS 1.18.6.4
Service 1.18.6.5
ENV 1.18.6.6
Install 1.18.6.7
Amazon ECS 1.18.7
底层技术 1.18.8
目录加载 1.18.8.1
为 RancherOS 做贡献 1.18.9
Rancher 配合Rancher OS 的使用技巧 1.18.10
常见问题 1.19
Rancher Server 1.19.1
Rancher Agent/Host 1.19.2
排错 1.19.3

Rancher 实战红宝书

Rancher 是非常优秀的容器管理平台，本书旨在沉淀和积累相关实战文档。本书通过 GitHub 和 GitBook 平台共同承载，并实现参与者的协作。本书使用开源社区的方式，由Rancher 用户社区合作维护。

参考源： http://docs.rancher.com

本书在线访问网址：http://rancher.hidocker.io

本书的初始化版本，保持了和 Rancher 文档官网同步的目录结构（为了维护方便，RancherOS 被当做一个独立章节纳入了本书）；为了方便您的翻译，请建议直接下载 https://github.com/rancher/rancher.github.io 英文源站到本地（作为英文原文的离线参考）；然后再 Fork https://github.com/martinliu/rancher-docs 到自己的仓库，并从自己的仓库下载本书电子书源码到本地，您会看到本书的很多章节还都是空白文章；然后您就可以可以开始工作了。本书的第一个阶段性目标是：对官方英文文档做 1：1的翻译。然后是收集各位高手的原创好文。目标是打造出一份很实用和全面的参考资料库。

如果你没有 GitHub 的使用经验，请查看这个视频，否则忽略此段，继续仔细阅读本网页；视频会教会您纯网页版的操作和协作流程，其中示例了如何把 6.1 访问控制 做英文原文的初始化，和翻译提交。视频点这里。http://v.youku.com/v_show/id_XMTU1NDk4Mzc2OA

Github 是一个协作平台，pull request 是解决冲突的方式，你提交的 pull request 上来以后，如果有冲突，在这种情况下电子书源码的管理员需要解决这个冲突，从两个冲突的 pull request 中选择出一个更好的。为了快速响应所有文档参与者的贡献，现在征召电子书源码仓库管理员合作者（Collaborators）；如果您提交的 PR 被三次接受，且您有意参与源码管理工作；您就可以申请成为 Collaborators ；请发一个 Issue 做出申请，没有人反对的话，2天内就会成为 Collaborators。

为了避免两个人或者更多的人同时开始翻译同一篇文章，当您开始决定做某个文章的时候，请通过首个 PR 来标记一下您的目标文章。具体的操作说明：打开您的目标文章（空白文章，只有一行标题的那种），在首行写一句话"本文档首稿正在编辑中，请随后帮忙review。"（初始化标记），提交一个更新 PR，等您的 PR 被确认之后，您即可进入了独占的编辑时段。您选择多篇文章的话，请在同一个 PR 中一次性提交初始化标记。

贡献者

本书源码开源托管在Github，欢迎参与维护：https://github.com/martinliu/rancher-docs， 贡献者名单见 GitHub contributors 页面。

欢迎参与线上讨论。



- QQ 群1
- 群名称：Rancher实战红宝书群1
- 群号：314700162

为了您能顺利领取到相关纪念品（首批为 Rancher logo 定制版 Tshirt），请加入以上 QQ 群，并用 Github ID 备注自己的姓名。 详情见 http://hidocker.io/book/

本书的结构

希望能形成一本书三卷的结构，系统而完整的介绍这个独特的容器管理平台。

第一部分

主要介绍 Rancher 自身的原理用法等，希望用户可以快速地把这款开箱即用的容器技术使用起来。它将覆盖 Rancher 管理平台服务器测的所有必要功能。

第二部分

主要介绍 RancherOS，这款专门为容器定制的精简 OS 的相关的内容。

第三部分

吸纳现有的各种实战文章，包括其它周边的配套话题。如容器平台的监控运维；基于此的 DevOps 的实践；在共有与中的最佳实践；与大数据或者私有云的相关话题。

参与协作

为了提高参与的效率，并省去在 Gitbook 上的操作的步骤。请直接加入本书在 GitHub 上的源代码协作来参与到文档的维护中来。

参与步骤：注册 Github 账号，克隆本书源码，下载到本机，编辑目标文章，推送到自己的仓库，提交 pull request 到本书源码，PR 通过之后，您的贡献自动跟新到线上版本。

在网页上就可以完成所有文章编辑和提交工作，操作流程在这里 http://v.youku.com/v_show/id_XMTU1NDk4Mzc2OA 网页版直接操作，请随时保存，以免网络中断导致的信息丢失。

具体操作步骤（Github 命令行版）

在 GitHub 上 fork 到自己的仓库，如 docker_user/rancher-docs，然后 clone 到本地，并设置用户信息。

```
$ git clone git@github.com:martinliu/rancher-docs.git
$ cd rancher-docs
$ git config user.name "yourname"
$ git config user.email "your email"
```

修改代码后提交，并推送到自己的仓库。

```
$ #do some change on the content 编辑您参与的文章
$ git commit -am "Fix issue #1: change helo to hello"
$ git push
```

在 GitHub 网站上提交 pull request。

定期使用项目仓库内容更新自己仓库内容。

```
$ git remote add upstream https://github.com/martinliu/rancher-docs
$ git fetch upstream
$ git checkout master
$ git rebase upstream/master
$ git push -f origin master
```

或者使用 GitHub Desktop 图形化客户端完成以上所有操作，推荐使用 Atom 作为本地 .mk 文件编辑器。

添砖加瓦

本书不仅限于对 Rancher 官方原文的翻译，更关注任何相关实战文档的汇聚。如果您有这个想法，请在 https://github.com/martinliu/rancher-docs/issues 页面点击 New Issue 按钮，在 Issue 中描述您想提交的“文章标题”和位置。Issue 被处理之后源码管理员会尽快初始化这篇文章，您同步更新到本地，就可以开始编辑上传这篇文章了。

主要版本历史

- v0.5.0 完成了第一版全目录，章节条目为 126 行，更新了 Readme
- v0.5.1 跟新了协作方法，用提交初始化标记 PR 替换了之前的认领方式，上传了网页版 github 操作视频。
-

Rancher 概述

Rancher 是以在生产环境中运行容器为目标而构建的开源软件平台。随着 Docker 容器这种类型的应用工作负载的逐渐流行，它催生了很多与之相应的基础架构服务，如网络服务、存储服务、负载均衡，安全，服务发现和资源管理。

计算资源

Rancher 使用的是来自于公有云或私有云上 Linux 主机的裸计算资源。每一个 Linux 主机既可以是虚拟机，也可以是物理机。Rancher 对每一个主机的期望不会多于 CPU，内存，磁盘存储和网络连接。从 Rancher 的角度看来，一个来自云服务商的云主机和私有数据中心的物理机是没多大差异。

关键功能

Rancher 产品的关键功能包括：

1. 扩主机网络： Rancher 为每个环境生成一个软件定义网络，为扩主机和云的容器之间提供了安全的网络通讯。
2. 容器的负载均衡。Rancher 提供的内置、弹性负载均衡能在容器之间或者服务之间分发流量。负载均衡服务可以跨多个云工作。
3. 持久化存储服务： Rancher 对 Docker 提供持久化存储服务的编排，让开发者在部署容器化应用的同时可靠地部署与之相应的存储。这项新功能基于 Docker 1.9的卷插件功能，这让开发人员可以更加方便地运行需要有状态数据库和持久存储的应用。
4. 服务发现：Rancher 实现了分布式服务发现功能，具有内置的健康检查功能，并使容器自动地注册自己到相应至相应服务，并且各种服务之间可以在网络上动态地彼此发现。
5. 服务升级：通过使用服务克隆和请求重定向功能，Rancher 使用户能更加容易地升级以及存在的容器服务。这让新版本的服务在处理生产流量前，有机会在其所依赖的生产环境中被校验和确认。
6. 资源管理：Rancher 支持 Docker Machine，这个强大的工具可以直接地对各种云提供商做主机部署。然后 Rancher 在对其做资源监控和容器部署管理。
7. 多租户和用户环境：Rancher 为多用户而设计，企业各个部门间可以跨应用生命周期协作。通过与已有目录服务的集成，Rancher 的用户可以创建独立的开发，测试和生产环境，然后邀请相关人员一起协作地管理资源和应用。

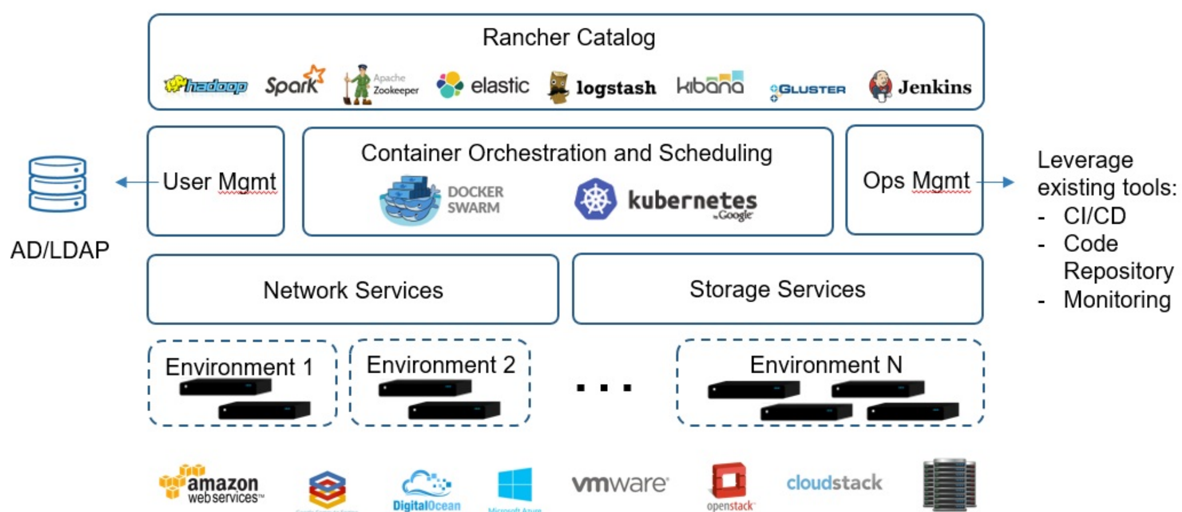
8. 多编排引擎支持：Rancher 用户在创建环境的时候，可以为他们的容器选择不同的容器编排引擎，默认是 Cattle，或者是 Kubernets 和 Docker Swarm。这让用户可以选择任意市场领先的调度框架的同时，依然能利用到Ranher 的其它所有功能，如：应用商店/目录，企业级用户管理，容器网络，和存储技术。

主要使用接口

用户有三种方式和 Rancher 交互：

1. 用户可以使用原生 Docker CLI 命令行或者 API 接口操作 Rancher 。Rancher 并不是其它那种把原生 Docker 体验给遮盖掉的编排系统或者管理系统。随着 Docker 平台不断地发展，封装层很可能会给原生的 Docker 功能给替代。而 Rancher 是在后台中工作，这样用户可以继续使用原生的 Dcoker CLI 和 Docker Compose 模板。Rancher 通过原生 Docker CLI 来用 Docker 标签的方式传递更多的信息，标签功能是 Rancher Labs贡献到 Docker 1.6中的功能。因为 Rancher 支持原生的 Docker CLI 和 API，像 Kubernets 这样第三方的工具可以在 Rancher 自如的使用。
2. 用户可以通过叫做 `rancher-compose` 的命令行工具和 rancher 交互。 `rancher-compose` 工具可以让用户基于 Docker Compose 模板在 Rancher 服务器上启动一套多容器和服务的应用。 `rancher-compose` 工具支持标准的 `docker-compose.yml` 文件格式。一个可选的 `rancher-compose.yml` 文件可以用来基于 `docker-compose.yml` 定义扩展或者覆盖原有的定义。
3. 用户还可以使用 Rancher 的图形界面来操作。Rancher图形界面可以完成很多配置工作，如配置访问控制权限，管理环境，添加 Docker 镜像库等。另外它为管理容器基础架构和服务提供了简单易用的用户体验。

Rancher 的主要功能如下的示意图，它可以被运行在任何云上，并且有三种方式与之交互。



本文档的线索

把 Rancher 运行起来是比较容易的。如果您有一个 Linux 虚拟机在笔记本上或者云主机，阅读 快速安装指南 迅速获得第一手实战经验。

如果您想安装一套生产级别的 Rancher 环境，请按照 Rancher 安装 这一章的指导，来部署配置 Rancher 服务器并添加节点进来。

在您开始使用 Rancher 之前，请一定阅读 基本概念 这一节，以了解 Rancher 是如何工作的。

系统配置这一章描述了大量 Rancher 服务器部署和运行之后，如何做各种必要的一次性配置工作。

这三章 --使用 Docker 原生命令行, 使用 Rancher Compose, 和 Rancher 图形界面-- 描述了使用 Rancher 功能的三种主要方式。

升级 Rancher 是很重要的一章，如果您在生产环境中运行 Rancher。

为Rancher做贡献 这一章包含了如何参与 Rancher 开源社区的信息。

Rancher OS 完整的这种定制版容器操作系统的安装、配置和使用，在英文原始文档中和本文为平行的一份文档。为了翻译协作和使用的方便性，目前把它收录为本书的一个章节。

快速安装指南

本文档首稿正在编辑中，请随后帮忙 review。

在本文中，我们将做一个 Rancher 系统的快速安装，即在一台 Linux 机器上安装并运行所有几乎所有的必要功能。

准备 Linux 主机

先安装一个64位的 Ubuntu 14.04 Linux 主机，其内核必须高于 3.10 。或者其它同等的 Linux 发行版。你可以使用一台笔记本、一个虚拟机或者一台物理的服务器。请确保目标安装 Linux 主机的内存至少1GB。

然后安装 Docker 在这个 Linux 主机上， 可以参考 Docker 网站的安装说明。

启动 Rancher 服务器

启动 Rancher 服务器所需要做的动作就只有一条命令。在启动了这个容器之后，我们将能查看到这个运行中的服务器的日志。

```
$ sudo docker run -d --restart=always -p 8080:8080 rancher/server
# 显示 Rancher 服务器的容器 ID，替换containerid
$ sudo docker ps
# 显示并查看 Rancher 服务器的日志
$ sudo docker logs -f containerid
```

启动 Rancher 服务器可能需要花几分钟时间。这取决于您下载 Rancher Server镜像的速度。当日志中显示 “.... Startup Succeeded, Listening on port...” 以后，Rancher UI 图形界面现在就能正常访问了。

Rancher 服务器的图形界面访问端口是 8080 ，通过在浏览器中访问这个网址 http://linux_host_ip:8080 , 您就可以打开 Rancher 服务器的图形界面。如果您的浏览器和 Rancher 服务器都运行在同一台服务器上，你需要使用主机的真实 Ip 地址，如：http://192.168.1.100:8080 ， 而不是 http://localhost:8080 或者 http://127.0.0.1:8080

> 注意: Rancher 的访问控制在初始安装时并没有配置，你的 Rancher 服务器图形界面和 API 能在任何能访问到您的 IP 地址的地方被访问到。我们建议配置访问控制参考 访问控制.

添加主机

为简化操作，我们将添加运行 Rancher 服务器容器的主机。而在实际的生产环境中，我们建议使用专用的主机来运行 Rancher 服务器。

通过点击图形界面的 **Infrastructure** 标签来添加主机，然后您将会看到 **Hosts** 页面。Rancher 会提示您选择一个 IP 地址。这个 IP 地址必须可以被所有即将添加的主机访问到。把 Rancher 服务器的端口通过防火墙的 NAT 或者负载均衡器暴露出来，或者暴露到 Internet 上在有些情况下是很有用的。如果你的主机有一个私有或者本地 IP 地址，例如：`192.168.*.*` ；Rancher 将打印一个提示信息，告诉您是否确认这个 IP 地址可以被正常访问到。

现在我们添加 Rancher 服务器主机自身，因此我们可以忽略这个提示信息。点击 **Save** ；您将进入默认的 **Custom** 选项页面，您在这可以得到运行 rancher/agent 容器的命令。这里还有其它的公有云的选项，使用这个选项可以实现通过 `docker-machine` 去启动主机节点。在 Web 界面上，Rancher 提供的用于添加主机的命令如下：

```
$ sudo docker run -d --privileged -v /var/run/docker.sock:/var/run/docker.sock rancher
/agent:v0.7.9 http://172.17.0.3:8080/v1/scripts/DB121CFBA836F9493653:1434085200000:2ZO
wUMd6fIzz44efikGhBP1veo
```

由于我们正在添加 Rancher 服务器的主机，我们需要添加这个主机所使用的共有 IP。Rancher agent 命令中如果没有这个参数，这个主机的 IP 很可能会是个错误的配置。您可以添加这个 IP 地址在**Step 4**，这将会修改命令，并加入一个环境变量。

```
$ sudo docker run -e CATTLE_AGENT_IP=172.17.0.3 -d --privileged -v /var/run/docker.soc
k:/var/run/docker.sock rancher/agent:v0.7.9 http://172.17.0.3:8080/v1/scripts/DB121CFB
A836F9493653:1434085200000:2ZOwUMd6fIzz44efikGhBP1veo
```

在运行 Rancher 服务器的主机上运行这个命令。

当您在 Rancher 的页面中点击 **Close** 按钮后，您会被返回到 **Infrastructure -> Hosts** 页面。在一两分钟后，这个主机将自动出现在这里。

使用图形界面创建一个容器

进入 **Applications -> Stacks** 页面，如果这里还没有服务，你可以点击 "Add Service" 按钮。你可以输入一个类似 “first*container*” 的名字。您现在使用默认配置并点击 ***Create*** 。*Rancher* 将开始在这个主机上启动两个容器。一个容器是您所创建的名为**_*first_container* ；另外一个容器是*Network Agent***，这是个由 Rancher 创建的系统容器，它用来处理扩主机联网和健康检查等任务。

不管你的主机是什么 IP 地址，***first_container*** 和 ***Network Agent*** 将会的到 `10.42.*.*` 网段的 IP 地址。Rancher 已经创建了能在不同主机之上的让所有容器可以相互通信的覆盖网络。

如果你点击 ***first_container***的下拉菜单，你可以执行各种动作，例如：停止容器，查看日志，或者进入容器的 控制台。

使用 Docker 原生命令创建一个容器

Rancher 会显示所有在主机上的容器，即使有些容器是在图形界面之外创建的。在主机的 shell 命令行里创建一个容器。

```
$ docker run -it --name=second_container ubuntu:14.04.2
```

在图形界面中，你将看到 ***second_container*** 在你的主机上出现！如果你通过退出命令行来退出用命令方式创建的容器，在 Rancher 图形界面中将立刻显示这个容器的状态为停止。

Rancher 可以对带外发生的事件作出反应，并把当前的显示状况如实地整合在它的视图中。你能了解更多信息在 使用 Docker 原生命令行 这一章。

如果你查看容器 ***second_container*** 的 IP 地址，你会注意到他不在 `10.42.*.*` 网段中。它的 IP 地址是通过 Docker 后台服务获得的。这是通过命令行方式创建容器的正常的结果。

如果我希望通过命令行创建的容器依然具有 Ranger 覆盖网络的网络地址呢？我们所需要做的就仅仅是在命令中加一个标签。

```
$ docker run -it --label io.rancher.container.network=true ubuntu:14.04.2
```

标签 `io.rancher.container.network` 让我们通过命令行传递了一个通知，这样 Rancher 会为把容器配置为连接到覆盖网络。

创建一个多容器应用

我们已经展示了如何创建当个容器，并把他们连接到跨主机的网络。然而大多数真实世界中的应用都是由多种服务构成的。例如一个 WordPress 应用是由下列服务组成的：

1. 一个负载均衡服务。负载均衡器把 Internet 流量转发给 WordPress 应用。
2. 一个由两个 WordPress 容器组成的 WordPress 服务。
3. 一个由一个 MySQL 容器组成的数据库服务。

负载均衡器的目标地是 WordPress 服务，WordPress 服务连接到 MySQL 服务。

在这一章里，我们将在 Rancher 中逐步地创建和部署 WordPress 应用堆栈。

在 Rancher 的图形界面中，点击 **Applications -> Stacks**，点击 **Add Service** 按钮来添加一个服务。

首先，我们将使用 mysql 镜像来创建一个名为 *database* 的数据库服务。在 **Command** 标签里，添加环境变量 `MYSQL_ROOT_PASSWORD=pass1` ，点击 **Create** 按钮。然后我们会立刻进入堆栈页面，这里包含了所有服务。

然后，再次点击 **Add Service** 按钮来添加另外一个服务。我们将添加一个 WordPress 服务并连接到 mysql 服务。让我们使用 *mywordpress* 做名称，使用 wordpress 镜像。我们将拉动滚动条增加此服务的容器数量到2。在 **Service Links** 标签中，添加 *database* 服务，并提供 *mysql* 名称。就想在 Docker 中一样，当你选择了 *mysql* 后，Rancher 将从所连接的数据库服务中连接必要的环境变量到 WordPress 镜像中。然后点击**Create**。

最后，我们来创建负载均衡器。点击 **Add Service**按钮傍边的下拉菜单。选择 **Add Load Balancer**。提供像 *wordpresslb* 这样的名称，并选择一个源端口和你将用来访问 wordpress 应用的主机上的目标端口。在这个例子中，我们把这两个端口都使用 `80` 。目标服务将是 *mywordpress* 。点击 **Save**。

至此我们的多服务应用堆栈创建完毕！在 **Applications -> Stack**页面，我们将能够在负载均衡器的开放端口处找到一个连接。点击这个连接，浏览器将新打开一个窗口，它将显示 wordpress 应用。

使用 Rancher Compose 创建一个多容器应用

在这一个部分，我们将使用名为 `rancher-compose` 的命令行工具来创建上一节里已经创建和部署的相同的 WordPress 应用。

命令行工具 `rancher-compose` 的功能和流行的 `docker-compose`命令行工具类似。它使用相同的`docker-compose.yml`文件来在 Rancher 上部署应用。你能在 `rancher-compose.yml` 文件中制定更多的属性，它会扩展和覆盖 `docker-compose.yml` 文件。

在上一节里，我们创建了一个具有一个负载均衡器的 WordPress 应用。如果你已经在 Ranher 中创建了它，你能直接在图形界面的堆栈的下拉菜单中选择 **Export Config** 来直接下载文件。`docker-compose.yml` 和 `rancher-compose.yml` 文件内容将与下面实例类似：

docker-compose.yml

```
mywordpress:
  tty: true
  image: wordpress
  links:
    database: mysql
  stdin_open: true
wordpresslb:
  ports:
  - 80:80
  tty: true
  image: rancher/load-balancer-service
  links:
    mywordpress: mywordpress
  stdin_open: true
database:
  environment:
    MYSQL_ROOT_PASSWORD: pass1
  tty: true
  image: mysql
  stdin_open: true
```

rancher-compose.yml

```
mywordpress:
  scale: 2
wordpresslb:
  scale: 1
  load_balancer_config:
    haproxy_config: {}
  health_check:
    port: 42
    interval: 2000
    unhealthy_threshold: 3
    healthy_threshold: 2
    response_timeout: 2000
database:
  scale: 1
```

从 Ranher 图形界面中点击 `Download CLI` 来下载 `rancher-compose` 可执行文件，这个链接位于页面的页脚。我们提供了 Windows，Mac 和 Linux 的不同的版本。

为了使用 `rancher-compose` 在 Rancher 中启动服务，你需要设置一些必须的变量。你需要在 Rancher 图形界面中创建一个 environment API Key 。点击 **API** ，在点击 **Add API Key**。保存用户名(access key)和密码(secret key)。设置 rancher-compose 所需要的环境变量： `RANCHER_URL` , `RANCHER_ACCESS_KEY` , 和 `RANCHER_SECRET_KEY` 。

```
# 设置 Rancher 的环境变量
$ export RANCHER_URL=http://server_ip:8080/
# 设置访问密钥，类似：username
$ export RANCHER_ACCESS_KEY=<username_of_key>
# 设置安全密钥，类似： password
$ export RANCHER_SECRET_KEY=<password_of_key>
```

现在进入保存 `docker-compose.yml` 和 `rancher-compose.yml` 文件的目录中，并运行下面的命令。

```
$ rancher-compose -p NewWordpress up
```

在 Rancher 中，一个名为 **NewWordPress** 的新堆栈将被创建，并且具有了所以的服务。

Rancher 安装

单节点服务器安装

Rancher 使用基于 Docker 容器的部署方式。仅需简单地启动两个容器即可运行。一个容器用于管理 Rancher 服务，其他容器使用代理的方式管理主机或节点。

需求

- 任何支持 Docker 1.10.3 的 Linux 发行版。其中 RancherOS，Ubuntu，RHEL/CentOS 7 经过了更多的测试
- 1GB 内存
- MySQL 服务，并且设置 max_connections > 150

启动 **Rancher** 服务器

在安装了 Docker 的服务器上可以很简单的使用命令启动 Rancher 服务器。

```
$ sudo docker run -d --restart=always -p 8080:8080 rancher/server
```

Rancher UI

Rancher 的 UI 和 API 默认工作在 8080 端口。Docker 镜像下载后，还需要 1-2 分钟才可以成功启动并显示页面。

访问以下地址：`http://<SERVER_IP>:8080` ，这里 `<SERVER_IP>` 是指运行 Rancher 服务器的主机用于网络访问的IP地址。

一旦 UI 处于启动和运行状态，就可以开始 添加主机 。主机添加到 Rancher 服务器中后，您可以开始添加 服务 或 使用 Rancher catalog 启动模板。

开启活动目录或 **OpenLDAP** 支持**(TLS)**

为了开启 Rancher 活动目录或 OpenLDAP 支持(TLS)，Rancher Server 容器在启动时需要加载证书。您需要在运行 Rancher Server 的 Linux 主机上保存证书。

启动 Rancher 服务器并使用绑定挂载卷的方式加载证书，证书在容器中 必须 使用 `ca.crt` 命名。

```
$ sudo docker run -d --restart=always -p 8080:8080 \
  -v /dir_that_contains_the_cert/cert.crt:/ca.crt rancher/server
```

您可以通过检查 Rancher 服务器容器日志确认 `ca.crt` 是否已经成功的传递给了 Rancher 服务器。

```
$ docker logs <server_container_id>
```

在日志的开始，会提示证书 `ca.crt` 已经添加。

```
DEFAULT_CATTLE_RANCHER_COMPOSE_WINDOWS_URL=https://releases.rancher.com/compose/beta/l
atest/rancher-compose-windows-386.zip
Adding ca.crt to Certs.
Updating certificates in /etc/ssl/certs... 1 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d....
done.
done.
[BOOTSTRAP] Starting Cattle
```

绑定挂载 MYSQL 卷

如果你想将容器内数据库持久化在主机上，可以在启动 Rancher 服务容器时，绑定并挂载 MySQL 数据卷。

```
$ sudo docker run -d -v <host_vol>:/var/lib/mysql --restart=always -p 8080:8080 ranche
r/server
```

使用这个命令，数据库将持久化保存在主机上。如果您已有一个 Rancher 服务器容器，请参照我们的 升级文档 。

使用外部数据库

如果您更希望使用外部的数据库运行 Rancher 服务器，请参照如下操作连接 Rancher 到数据库。您需要一个已经创建好的数据库，但是不需要创建任何数据库对象，Rancher 将会自动创建所有相关的数据库对象。

Rancher 服务器需要使用以下环境变量，并通过 `docker run` 命令启动 Rancher 服务器来连接外部数据库。

- CATTLE_DB_CATTLE_MYSQL_HOST: 数据库实例的主机名或IP地址
- CATTLE_DB_CATTLE_MYSQL_PORT: 3306
- CATTLE_DB_CATTLE_MYSQL_NAME: 数据库名

- CATTLE_DB_CATTLE_USERNAME: 用户名
- CATTLE_DB_CATTLE_PASSWORD: 密码

> 注意：
>
> 数据库名和用户名必须已经存在，Rancher 不会去创建数据库。

这是一个用于创建数据库和用户的 SQL 命令。

```
sql CREATE DATABASE IF NOT EXISTS cattle COLLATE = 'utf8_general_ci' CHARACTER SET = 'utf8'; GRANT ALL ON cattle.* TO 'cattle'@'%' IDENTIFIED BY 'cattle'; GRANT ALL ON cattle.* TO 'cattle'@'localhost' IDENTIFIED BY 'cattle';
```

以上命令创建了名为 'cattle' 的数据库，创建了用户名为 'cattle' 的用户。

创建了数据库和用户后，使用环境变量参数，并启动 Rancher 服务器。

```
$ sudo docker run -d --restart=always -p 8080:8080 \
    -e CATTLE_DB_CATTLE_MYSQL_HOST=<hostname or IP of MySQL instance> \
    -e CATTLE_DB_CATTLE_MYSQL_PORT=<port> \
    -e CATTLE_DB_CATTLE_MYSQL_NAME=<Name of Database> \
    -e CATTLE_DB_CATTLE_USERNAME=<Username> \
    -e CATTLE_DB_CATTLE_PASSWORD=<Password> \
    rancher/server
```

使用 HTTP 代理环境变量

为了使用 HTTP 代理，需要修改 Docker 服务支持代理。在 Rancher 服务器启动前，编辑 `/etc/defalut/docker` 文件指定您的代理并重新启动 Docker 服务。

```
$ sudo vi /etc/default/docker
```

在这个文件中编辑 `#export http_proxy="http://127.0.0.1:3128/"` 指向您的代理。保存修改并重新启动Docker 服务，每个操作系统重启 Docker 服务的方式是不同的。

> 注意：
>
> 如果您使用 systemd 运行 Docker ，请参照 Docker 文档 配置 HTTP 代理。

为了加载 Rancher catalog ，需要在启动 Rancher 服务器时加载 HTTP 代理的环境变量信息。

```
$ sudo docker run -d \
    -e http_proxy=<proxyURL> \
    -e https_proxy=<proxyURL> \
    -e no_proxy="localhost,127.0.0.1" \
    --restart=always -p 8080:8080 rancher/server
```

如果不使用 Rancher catalog ，则使用正常方式运行启动 Rancher Server 命令。

在 Rancher 中 添加主机 ，无需使用 HTTP 代理。

> 译者： XiaoBao Zhang
>
> From： http://docs.rancher.com/rancher/installing-rancher/installing-server/

多节点高可用(HA)

基于 **Rancher v1.0.1**

基本配置需求

- 多节点的HA配置请参照单节点 需求
  - 节点需要开放的端口
    - 全局访问：TCP 端口 `22`，`80`，`443`，`18080`（可选：用于在集群启动前 查看并管理栈）
    - 节点间连接：
      - UDP 端口：`500`，`4500`
      - TCP 端口：`2181`，`2376`，`2888`，`3888`，`6379`
- MySQL 数据库
  - 至少 1GB 内存
  - 每 Rancher 服务器节点 50 个连接（如：3 节点的高可用部署至少需要支持 150 个连接）
- 外部负载均衡器

建议配置

- 每个 Rancher 服务器节点应该有 4GB 或 8GB 可用内存，意味着至少需要 8GB 或 16GB 的物理内存
- MySQL 数据库应该使用高速的磁盘
- 对于真正的高可用，建议使用主从的 MySQL，并做适当的备份。由于存在事物锁，可以选择 Galera 或强制写入单节点的方式。

配置高可用的准备

1. 根据 使用外部数据库启动单节点 说明部署一个至少拥有 1GB 内存的 MySQL 数据库，但是不要使用其中启动 Rancher 服务器的相关指令。因为默认情况下， 用户只能从本地访问数据库，你需要授权所有 Rancher 服务器节点对其的网络访问。
2. 配置一个外部负载均衡器并将端口 80 和 443 的流量指向运行 Rancher Server 的节点池。
3. 使用本文准备所有节点。这些节点都应该满足单节点部署 Rancher 服务器的需求。（可选）您可以提前拉取 `rancher/server` 镜像到这些节点上。

   目前，我们的高可用集群支持 3 种配置。 1 节点：没有高可用； 3节点：任何一台主机可以宕机； 5 节点：任何两台主机可以宕机。

> 注意：
>
> 这些节点可以分布在同一个地区，并使用稳定的高速链路连接的多个数据中心，不建议分布在距离较远的区域中。如果你选择分布节点在同一个区域，Zookeeper 可以用来保证集群的高可用。如果你的节点分布在不同的数据中心，那么你只能保留问题最少的那个区域。

4. 在其中一个节点上，启动一个 Rancher 服务器用于生成配置脚本。下面这个脚本用于生成 Rancher 服务器同时连接到外部数据库并初始化数据。它将被引导高可用部署过程。最终，Rancher 服务器容器将使用此步骤替换为支持高可用的 Rancher 服务器容器。

```
$ sudo docker run -d -p 8080:8080 \
-e CATTLE_DB_CATTLE_MYSQL_HOST=<hostname or IP of MySQL instance> \
-e CATTLE_DB_CATTLE_MYSQL_PORT=<port> \
-e CATTLE_DB_CATTLE_MYSQL_NAME=<Name of Database> \
-e CATTLE_DB_CATTLE_USERNAME=<Username> \
-e CATTLE_DB_CATTLE_PASSWORD=<Password> \
-v /var/run/docker.sock:/var/run/docker.sock \
rancher/server:v1.0.1
```

> 注意：
>
> 请耐心等待，这个初始化步骤可能要15分钟才能完成

5. （可选）预先下载 `rancher/server` 镜像到 Rancher 节点。这里下载的镜像可用于配置脚本生成 Rancher 服务器。

```
# 这个版本需要是第四步中所使用的版本
$ sudo docker pull rancher/server:v1.0.1
```

生成配置脚本

1. 访问 Rancher 服务器地址 `http://<server_IP>:8080` 生成脚本。在 **Admin -> HA** 确认 Rancher 服务器已经成功连接到外部数据库。如果没有正确配置，请重复上一节中的步骤 1 和 4 。
2. 选择集群大小，应该为您的 Rancher 服务器节点数量，参照上一节中步骤 3 。
3. 在 **Host Registration URL** 中填写外部负载均衡器的 IPv4 地址或主机名。
4. 选择您想使用的证书类型。Rancher 服务器可以为您生成一个自签名证书或者使用自己的有效证书。
5. 点击 **Generate Config Script** 。
6. 下载脚本并保存到本地。
7. 保存脚本后，停止用于生成脚本的 Rancher 服务器容器。

启动Rancher高可用

1. 为了使所有节点支持高可用，你需要在所有节点上使用配置脚本启动 Rancher 服务器 。脚本将启动一个 Rancher 服务器容器并连接到之前创建的外部数据库。

   > 注意：
   >
   > 请确保您已经停止用于生成脚本 `rancher-ha.sh` 的 Rancher 服务器容器后再运行配置脚本。否则，在你尝试在同一个节点运行配置脚本时，将会有一个端口冲突导致高可用节点无法启动。

2. 如果你之前生成配置脚本时提供了 **Host Registration URL** ，请导航到外部负载均衡器的 IP 或主机名。请注意，Rancher 服务器的用户界面可能需要几分钟才可以使用。如果你的用户界面仍不可用，请参照 查看并管理栈 。

3. 一旦用户界面可用，您将可以添加主机到 Rancher 高可用集群。在 **Admin -> HA** 标签可以查看高可用节点的数量。添加主机前，您需要保存证书 `/var/lib/rancher/etc/ssl/ca.crt` 并赋予 `400` 权限到您要添加的主机上。注册命令可以自动创建使用并管理证书。
4. 在您向环境中添加主机后，高可用设置已经完成，您可以开始通过用户界面 添加服务 ，从目录 启动模板 或使用 rancher-compose 启动服务。

   > 注意：
   >
   > 如果您正在使用 AWS ，你需要为添加到 Rancher 的主机配置 IP 。如果你想添加 自定义主机 ，你需要在配置页面中填写公网 IP ，启动 Rancher agent 的命令会相应改变。已经通过页面添加的主机，必须 ssh 登陆到机器重启 Rancher agent 使 IP 生效。

> 译者： XiaoBao Zhang
>
> From： http://docs.rancher.com/rancher/installing-rancher/installing-server/multi-nodes/

SSL基本配置

为了可以正常使用 `https` 地址访问 Rancher 服务器，您需要设置代理的证书卸载功能。同时，我们提供了一些配置 NGINX 或 Apache 代理的例子，当然，您也可以使用其他工具。

需求

除了典型的 Rancher 服务器 需求 您还需要：

- 有效的 SSL 证书：如果您的证书不包含在 Ubuntu CA 中，请参照 自签名证书说明 。
- 配置 DNS 记录。

启动 Rancher 服务器

在我们的示例配置方案中，所有流量将通过代理被发送到 Rancher 服务器的 Docker 容器中。也有替代的配置方案可以实现，但这个示例相对比较简单。 启动 Rancher 服务器，这里我们添加了选项 `--name=rancher-server` 用来连接代理容器和 Rancher 服务器容器。

```
$ sudo docker run -d --restart=always --name=rancher-server rancher/server
```

> 注意：
>
> 在我们的示例中，我们假设代理运行在单独的容器中。如果您的代理计划由主机提供服务，需要将 Rancher 容器的 `8080` 端口映射至本地主机，可以向 `docker run` 命令添加参数 `-p 127.0.0.1:8080:8080` 实现。

如果您想使用现有的 Rancher 实例，您可以基于现有的 Rancher 实例创建一个新的 Rancher 实例。

- 若 Rancher 实例的 MySQL 数据库运行在容器内，在启动新的 Rancher 实例时，请参照 升级说明 创建数据容器并且在启动时添加 `--volumes-from=<data_container>` 。
- 若 Rancher 实例使用了 绑定挂载数据库 的方式，请参照 绑定挂载实例升级说明。
- 若 Rancher 实例使用外部数据库，则直接停止并删除现有的 Rancher 实例容器，并参照 外部数据库连接说明 启动新的容器。

> 注意：
>
> 在新的 Rancher 实例容器运行后，请确认已经 删除 了旧的 Rancher 实例容器，否则，在您的主机重启后，因为我们在 `docker run` 命令时加入了 `--restart=always` 参数，会导致旧容器启动。

NGINX 配置示例

这里的配置是 NGINX 运行的最低配置，您应该定制配置来满足您的需求。

节点设置

- `rancher-server` 是您的 Rancher 服务器容器名。您必须在启动 Rancher 服务器容器的时候加入 `--name=rancher-server` 选项。并且在启动 NGINX 容器的时候加入 `--link=rancher-server` 选项才能使这些配置正常工作。
- `<server>` 可以任意命名，但是必须保证 http 和 https 配置中的名称相同。

```
upstream rancher {
    server rancher-server:8080;
}

server {
    listen 443 ssl;
    server_name <server>;
    ssl_certificate <cert_file>;
    ssl_certificate_key <key_file>;

    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Port $server_port;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://rancher;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        # This allows the ability for the execute shell window to remain open for up to 15 minutes. Without this parameter, the default is 1 minute and will automatically close.
        proxy_read_timeout 900s;
    }
}

server {
    listen 80;
    server_name <server>;
    return 301 https://$server_name$request_uri;
}
```

APACHE 配置示例

这是一个 Apache 的配置。

节点设置

- `<server_name>` 是您 Rancher 服务器容器的名字，所以在启动 Apache 容器时需要加入 `--link=<server_name>` 才能使配置正常工作。
- 在代理配置中，您需要替换 `rancher` 为您的信息。

```
<VirtualHost *:80>
  ServerName <server_name>
  Redirect / https://<server_name>/
</VirtualHost>

<VirtualHost *:443>
  ServerName <server_name>

  SSLEngine on
  SSLCertificateFile </path/to/ssl/cert_file>
  SSLCertificateKeyFile </path/to/ssl/key_file>

  ProxyRequests Off
  ProxyPreserveHost On

  RewriteEngine On
  RewriteCond %{HTTP:Connection} Upgrade [NC]
  RewriteCond %{HTTP:Upgrade} websocket [NC]
  RewriteRule /(.*) ws://rancher:8080/$1 [P,L]

  RequestHeader set X-Forwarded-Proto "https"
  RequestHeader set X-Forwarded-Port "443"

  <Location />
    ProxyPass "http://rancher:8080/"
    ProxyPassReverse "http://rancher:8080/"
  </Location>

</VirtualHost>
```

更新主机注册

在 Rancher 配置完成后，用户界面将使用 `https://<your domain>/` 访问。在 添加主机 之前，您需要正确配置 SSL 方式的 主机注册 地址。

在 AWS ELB 后使用 SSL 运行 Rancher

默认情况下，ELB 不支持在 HTTP/HTTPS 模式下启用 websockets 。由于 Rancher 使用了 websockets ，则 ELB 必须修改配置以便 Rancher 的 websockets 可以正常工作。

Rancher 的 ELB 配置需求

- 启用 代理协议 模式
- 配置 TLS/SSL 方式的前端以及 TCP 方式的后端

使用自签名证书（测试）

免责声明

这个配置可以工作在单节点 Rancher 服务器（非HA安装）模式的核心服务上，没有人验证是否支持 Rancher 目录 是否被支持。

Rancher Compose 命令行默认需要将 CA 证书存储在操作系统，请参照 Golang。

服务器必备条件

- CA 证书 PEM 格式的文件
- 已经签署证书的 Rancher 服务器
- NGINX 和 Apache 配置证书卸载，并反向代理到 Rancher 服务器

Rancher 服务器

1. 使用修改后的 Docker 命令启动 Rancher 服务器容器。证书 必须 在容器中被命名为 `ca.crt` 。

   ```
   $ sudo docker run -d --restart=always -p 8080:8080 -v /some/dir/cert.crt:/ca.crt
   rancher/server
   ```

   > 注意：
   >
   > 如果你正在运行 NGINX 或 Apache 容器，您可以直接访问这些实例，不需要通过 Rancher 服务器的 8080 端口访问用户界面。 这条命令将会配置 Rancher 服务器的证书链，所以 Rancher 的服务，如机器配置，目录和 compose 均可以和 Rancher 服务器通讯。

2. 如果您正在使用一个 NGINX 或 Apache 容器卸载证书，您需要在启动容器的命令增加 `--link=` 选项。
3. 通过 `https` 访问 Rancher 用户界面，即 `https://rancher.server.domain` 。
4. 更新 主机注册 为 SSL 方式。

> 注意：
>
> 除非您的机器信任了用于签署 Rancher 服务器的 CA 证书，否则当你访问页面时，浏览器会给出一个不可信网站的警告信息。

添加主机

1. 如果您想添加主机到 Rancher ，您必须在主机上以 pem 格式将 CA 证书保存到 `/var/lib/rancher/etc/ssl` 目录并命名为 `ca.crt` 。
2. 添加 自定义主机 ，也就是从用户界面粘贴命令。命令已经加入了 `-v /var/lib/rancher:/var/lib/rancher` 选项，所以文件会被自动复制到您的主机上。

> 译者： XiaoBao Zhang
>
> From： http://docs.rancher.com/rancher/latest/en/installing-rancher/installing-server/basic-ssl-config/

无互联网访问

Rancher 服务器的运行可以不需要互联网，浏览器需要使用内网地址来访问 Rancher 的图形界面。Ranher 可以配置私有的镜像仓库或者 HTTP 代理。

在没有互联网访问的网络中运行 Rancher 服务器有下面几个功能将无法工作正常。

- 从网页界面上启动容器云主机 - 由于 Rancher 是通过调用 `docker-machine` 来在公有云里创建主机，因此这个功能会不能用。您依然可以使用自定义主机来添加主机。
- GitHub 用户认证。
- 来自 Rancher Catalog 和 Community Catalog 的模板 - 这些目录依赖从 Github 上做克隆，但是您将仍然可以使用 添加内部目录 来把私有目录添加到 Rancher 中。

使用私有镜像仓库

这里假设您已经有了自己的内部私有镜像仓库或者相关方式来做 docker 镜像的分发。如果您在搭建私有镜像仓库方面需要帮助，请参考 Docker 官方文档 Docker documentation for private registries.

Push 镜像到私有仓库

在您开始安装或者升级 Rancher 服务器之前，确保所有镜像(例如： rancher/server, rancher/agent, rancher/agent-instance) 都已经分发到所有服务器是 非常重要的 如果您的私有镜像仓库中没有这些镜像，Rancher 服务器将可能变得不稳定。

对与 Rancher 服务器的每个版本，它所需要的相应的 Rancher agent 和 Rancher agent instance 镜像的版本将会在 Rancher 的发行说明文档中。

推送镜像到私有仓库的命令

下面的例子适用于 v1.0.1 版本的 Rancher 服务器去访问 DockerHub 和您的私有仓库。我们建议在私有镜像仓库中的镜像打上相同的版本和标签。

```
# rancher/server
$ docker pull rancher/server:v1.0.1
$ docker tag rancher/server:v1.0.1 localhost:5000/<NAME_OF_LOCAL_RANCHER_SERVER_IMAGE>
:v1.0.1
$ docker push localhost:5000/<NAME_OF_LOCAL_RANCHER_SERVER_IMAGE>:v1.0.1

# rancher/agent
$ docker pull rancher/agent:v1.0.1
$ docker tag rancher/agent:v1.0.1 localhost:5000/<NAME_OF_LOCAL_RANCHER_AGENT_IMAGE>:v
1.0.1
$ docker push localhost:5000/<NAME_OF_LOCAL_RANCHER_AGENT_IMAGE>:v1.0.1

# rancher/agent-instance
$ docker pull rancher/agent-instance:v0.8.1
$ docker tag rancher/agent-instance:v0.8.1 localhost:5000/<NAME_OF_LOCAL_RANCHER_AGENT
_INSTANCE_IMAGE>:v0.8.1
$ docker push localhost:5000/<NAME_OF_LOCAL_RANCHER_AGENT_INSTANCE_IMAGE>:v0.8.1
```

从私有镜像仓库启动 Rancher 服务器

在您的服务器上，使用您特定的 Rancher 镜像启动 Rancher 服务器。我们使用特定的版本号标签，而不是 `latest` 标签，这样能确保您工作在正确的版本组合上。

使用 v1.0.1 版本示例：

```
$ sudo docker run -d --restart=always -p 8080:8080 \
    -e CATTLE_BOOTSTRAP_REQUIRED_IMAGE=<Private_Registry_Domain>:5000/<NAME_OF_LOCAL_R
ANCHER_AGENT_IMAGE>:v1.0.1 \
    -e CATTLE_AGENT_INSTANCE_IMAGE=<Private_Registry_Domain>:5000/<NAME_OF_LOCAL_RANCH
ER_AGENT_INSTANCE_IMAGE>:v0.8.1 \
    <Private_Registry_Domain>:5000/<NAME_OF_LOCAL_RANCHER_SERVER_IMAGE>:v1.0.1
```

Rancher 图形界面

图形界面和 API 的访问是通过暴露的 `8080` 端口。您可以在浏览器中访问这个 URL: `http://<SERVER_IP>:8080` .

添加主机

在进入图形界面之后，点击 **Add Host** 按钮。 然后 **Host Registration** 页面会立刻显示出来，点击 **Save**。

这里使用 `docker-machine` 添加公有云主机是不能工作的。点击 **Custom** 图标来获取添加主机的命令。

网页中的命令将被配置为使用私有镜像仓库去拉取 Rancher agent 镜像。

添加自定义主机命令示例

```
$ sudo docker run -d --privileged -v /var/run/docker.sock:/var/run/docker.sock <Privat
e_Registry_Domain>:5000/<NAME_OF_LOCAL_RANCHER_AGENT_IMAGE>:v1.0.1 http://<SERVER_IP>:
8080/v1/scripts/<security_credentials>
```

使用 HTTP 代理

注意，在这个安装中浏览器是通过访问私有网络来使用 Rancher 的图形界面。

配置 Docker 使用 HTTP 代理

为了配置 HTTP 代理，Docker 后台进程在被修改配置之后可以使 Rancher 服务器和 Rancher 主机指向 HTTP 代理。在启动 Rancher 服务器和 Rancher Agent 之前，编辑配置文件 `/etc/default/docker` 添加你您的代理服务器，并重启 Docker 服务器。

```
$ sudo vi /etc/default/docker
```

在这个文件中，编辑 `#export http_proxy="http://127.0.0.1:3128/"` 这个参数去指向您的代理服务器。保持变更并重启 Docker 服务。重启 Docker 容器服务器在不同 Linux 发行版上可能命令不同。

> 注意: 如果你使用 systemd 运行 Docker 服务，请参考 Docker 官方的关于配置 HTTP 代理服务器的文档 instructions 。

启动 Ranher 服务器

当使用代理服务器的时候，Rancher 服务器的启动时不需要加什么特别的环境变量参数的。因此，启动 Rancher 服务器容器的命令还是常规的格式。

```
sudo docker run -d --restart=always -p 8080:8080 rancher/server
```

Rancher 图形界面

图形界面和 API 的访问是通过暴露的 `8080` 端口。您可以在浏览器中访问这个 URL: `http://<SERVER_IP>:8080` 。

添加主机

在进入图形界面之后，点击 **Add Host** 按钮。 然后 **Host Registration** 页面会立刻显示出来，点击 **Save**。

这里使用 `docker-machine` 添加公有云主机是不能工作的。点击 **Custom** 图标来获取添加主机的命令。

网页上所显示的命令可以用于添加任何一台已经配置好Docker 使用 HTTP 代理的主机。

添加主机

在 Rancher 中，我们在网页上提供了关于如何添加公有云主机的简洁的说明，还包括了对于不支持的公有云如何做的说明。在 **Infrastructure** 页面点击 **Hosts** 菜单，在点击 **Add Host** 按钮。

主机最小化需求

- 任何能支持 Docker 1.10.3 的 Linux 发行版。其中 RancherOS, Ubuntu, RHEL/CentOS 7 是经过了大量测试的。
- 1GB 内存
- 建议 CPU 具备 AES-NI 特性

工作机制?

当 Rancher Agent 容器在主机上被启动了之后，主机开始去连接 Rancher 服务器。在**Add Host** -> **Custom** 页面上显示的很长的添加主机命令中的注册秘钥被 Rancher Agent 用来做对服务器端的首次连接。基于这个连接，它在 Rancher 服务器中生成了一个 Agent 账户和 API 密钥对。这个秘钥对将被用于后续所有与服务器端的通讯中，这种认证方式和环境 API 秘钥的认证鉴权逻辑是完全一致的。

这种设计是假设运行在外部的主机和硬件是不可信的，由于这些资源具有潜在的风险。Agent 的账号仅仅具有它所需要访问的资源的权限，由 Agent 端发送来的事件实际上是被检查的。而反方向并没有Agent 去校验主机的机制，因此您还可以配置用于 Agent 和 Rancher 服务器通讯的 TLS 证书校验。

不同环境的 IPSec 密钥是不同的，它被保存在数据库里，它在 Agent 用 API 密钥对注册的时候被发送给主机。主机直接的连接是点对点的，是 AES 加密的，这种加密是可以被大多数型号的 CPU 做加速的。

添加主机

在第一次做主机添加的时候，你需要配置一次 Host Registration。这个配置决定了主机用什么 DNS 名称或者 IP 地址和端口来连接 Rancher API。默认，我们已经选择了管理 IP 和端口 `8080` 。如果您确定需要改变 IP 地址，请确认这个地址和端口是被用来连接 RAncher API 的。在任何时候，您能更新Host Registration。在完成主机注册的配置后，点击 **Save** 按钮。

我们支持直接添加公有云主机或者云里已经创建好的主机。对于公有云，我们通过 `docker-machine` 创建主机，并支持任何支持 `docker-machine` 的云主机镜像。

选择你想添加的公有云主机：

- Adding Custom Hosts
- Adding Amazon EC2 Hosts
- Adding Azure Hosts
- Adding DigitalOcean Hosts
- Adding Exoscale Hosts
- Adding Packet Hosts
- Adding Rackspace Hosts
- Adding Hosts from Other Drivers

当一个主机被添加到 Rancher 中，一个 Rancher agent 容器在该主机上被启动。Ranher 将自动地下拉正确版本号的 `rancher/agent` 镜像，来运行所需版本的 Agent。Agent 版本的标签必须对应用相应的 Rancher 服务器版本。

主机标签

对于每一个主机，你可以通过打标签的方式来组织它们。当启动 rancher/agent 容器的时候，标签被当做环境变量来添加了。在图形界面中添加主机标签是用键/值对格式的，其中见必须是唯一标识符。如果你添加了两个相同的键，并使用不了不同的值，我们将只使用后一个输入的键/值对。

通过给主机打标签，你可以在之后的服务调度/负载均衡/服务定义 (schedule services/load balancers/services)中用到，如给某个服务 services 建立一个它运行主机的白名单或者黑名单。

如果您打算使用一个外部的 DNS 服务 external DNS service 那么您需要让 DNS 记录使用一个 IP 而不是主机 IP to program the DNS records using an IP other than the host IP，接着您需要在主机上使用这个标签 `io.rancher.host.external_dns_ip=<IP_TO_BE_USED_FOR_EXTERNAL_DNS>` 。打此主机标签可以在注册这个主机的时候或者主机已经被添加到 Ranher 中以后，它应该在外部 DNS 服务开始使用到之前做。这个标签的值对于外部的 DNS 服务需要是符合某种可编程的逻辑。

当使用图形界面添加不同类型的云主机的时候，rancher/agent 命令会被自动地启动，并会使用网页图形界面中您所制定的标签。

当您添加自动定义主机是，您可以在图形界面中添加标签，并且它会自动地添加环境变量(`CATTLE_HOST_LABELS`) 标签对到命令中。

示例

```
# 在 rancher/agent 命令中添加一个主机标签
$  sudo docker run -e CATTLE_HOST_LABELS='foo=bar' -d --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock rancher/agent:v0.8.2 \
    http://<rancher-server-ip>:8080/v1/projects/1a5/scripts/<registrationToken>

# 添加多个主机标签需要使用`&`来连接在一起
$  sudo docker run -e CATTLE_HOST_LABELS='foo=bar&hello=world' -d --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock rancher/agent:v0.8.2 \
    http://<rancher-server-ip>:8080/v1/projects/1a5/scripts/<registrationToken>
```

> 注意: `rancher/agent` 的版本是必须和 Rancher 服务器版本保持对应的。你需要在这里检查自定义命令中所使用的版本号标签是正确的。

自动化添加的主机标签

Rancher 自动化生成的主机标签是关于主机的 Linux 内核版本和 Docker 引擎版本的。

键	值	描述
`io.rancher.host.linux_kernel_version`	主机的 Linux 内核版本 (例如： `3.19`)	主机所运行的 Linux 内核版本号。
`io.rancher.host.docker_version`	主机的 Docker 版本 (例如： `1.10.3`)	主机所安装的 Docker 引擎版本。

主机在代理服务器后面

为了支持主机位于代理服务器之后，您将修改 Docker 引擎的配置来指向一个代理服务器。对此的描述请见 adding custom host page页面。

登录云主机

在 Rancher 启动主机之后，您可以登录这些主机。我们提供在创建主机过程中生成的证书，并提供下载。点击主机上的**Machine Config** 下拉菜单。这将下载一个包含所有证书的 tar.gz 文件。

SSH 登录到您的主机，打开命令行工具，进入包含所有证书文件的目录，使用 `id_rsa` 证书，用如下命令连接。

```
$ ssh -i id_rsa root@<IP_OF_HOST>
```

克隆主机

由于需要使用一个访问密钥来启动云主机，您可能想通过不用每次都输入密钥的方式来更简单地创建另外其他的云主机。Rancher 提供了克隆的功能来创建新相似的云主机。从主机的下拉菜单中选择 **Clone** 。它会带您进入**Add Host**页面，这个页面上的密码部分已经为您填写好了。

编辑主机

这个菜单位于一个已经添加成功的主机的下拉菜单中。在 **Infrastructure -> Hosts**页面，该菜单当您的鼠标滑过主机的下拉菜单按钮的时候出现。如果您点击主机名来查看主机详情，此菜单在页面的右上角的下拉菜单图标中。它就位于主机状旁。

如果您点击 **Edit** 菜单，您可以更新名称，描述或者主机的标签。

停用/激活主机

停用主机将把主机置于 *Inactive* 状态。在这个状态里，不会有新容器被部署。该主机上任何已经存在的活动的容器请依然照旧运行，对容器还是可以执行动作（启动/停止/重启）。这个主机将会继续保持和Rancher 服务器的连接。

当一个主机处于*Inactive* 状态，您还可以通过点击主机的下拉菜单中的 **Activate** 使其返回到 *Active* 状态。

> 注意: 如果一个主机在 Rancher 中已经宕机（例如：处于 `reconnecting` 或 `inactive` 状态），您将需要执行一次健康检查来让您的服务所使用的容器在另外的主机上启动起来。

删除主机

为了从服务器上删除一个主机，您将需要在主机的下拉菜单中做这些操作。

选择 **Deactivate** 。当主机已经完全处于停用状态了，主机将进入 *Inactive* 状态。选择 **Delete**。服务器将开始执行主机删除的操作。在完成了删除动作之后状态先会变为*Removed*。在彻底重图形界面上消失之前它还会进入 *Purged* 状态。

如果主机是在 Ranher 中创建的云主机，主机会将被从云端删除。如果云主机是通过使用自定义命令添加的，主机将依然保留在云端。

> 注意: 对于自定义主机，所有容器包括 Rancher agent 容器都还会保留在主机上。

在 Rancher 之外删除主机

如果主机不是在 Rancher 中删除的，Rancher 服务器还会持续连接它。最终，这个主机的状态会持续为 *Reconnecting* 状态，即使它没有被重连成功。因此您需要在图形界面中通过 **Delete** 删除这些主机，并它们清除掉。

基本概念

在本章节，我们将介绍Rancher的基本概念。在使用Rancher之前，我们建议你对这些概念应该非常熟悉。

用户

“用户”定义了什么角色在一个环境里能够拥有查看或管理Rancher资源的权限。Rancher缺省会允许单一租户的访问权限，当然，管理员可以定义多个用户的访问权限。

在你开启鉴权（authenticaiton）之前，可参照 访问控制 .

环境

所有的主机和任意的Rancher资源，例如容器、负载均衡器等，都创建并归属于一个环境。用户对这些资源查看和管理的权限，由环境的拥有者来定义。Rancher目前支持每个用户来管理和邀请其它用户进入他的环境，并且能够为多种工作负载来定义不同的环境。 例如，你可以创建一个“Dev”环境， 另外再创建一个与之隔离的“生产”环境，每个环境里定义不同的资源，从而限制不同用户对不同环境的访问权限。

在你希望和其它用户 共享环境 之前，请设置访问控制。

主机

在rancher环境里，主机是一个最基本的资源单元，可以是任意的虚拟或物理的Linux服务器，对这些服务器的要求如下:

- 任意的Linux 发行版，只要它支持Docker 1.10.3。
- 能够和Rancher服务器通过预定义的端口号，一般是8080， 以http或https的协议进行通讯。
- 和同环境里的其它主机路由可达，从而能够利用Rancher的跨节点网络进行通讯。

Rancher也支持Docker Machine命令，从而允许你通过任意Rancher支持的接口来添加主机。

在Rancher环境里添加主机操作之前，可参考添加第一台主机。

网络

Rancher通过使用IPsec隧道技术，构建了一个简单和安全的覆盖（Overlay）网络，来支持跨主机节点的容器通讯。如果希望使用该网络，容器在通过Rancher启动时，必须将其网络模式设为“Managed”，或着在Docker命令行启动时，提供额外的标签 "--label io.rancher.container.network=true"。大多数的Rancher网络功能，例如负载均衡或DNS服务，都需要容器设为“Managed”网络模式。

在Rancher网络里， 一个容器将被分配一个Docker的桥接IP地址(172.17.0.0/16) 和一个Rancher管理的IP地址 (10.42.0.0/16) ，该地址配置在缺省的docker0桥上. 通过这样的设置，在同一个环境里的所有容器将都能够通过“managed”网络实现路由可达和互相访问。

备注*:* _Rancher的管理IP地址在Docker meta-data里是不存在的，所以通过Docker的“inspect”指令是看不到的。有些时候这一点可能会和一些需要Docker桥接IP地址的工具不兼容。我们正在和Docker社区共同合作，确保Docker的未来版本更清晰地支持Overlay网络。

服务发现

Rancher采用标准的Docker Compose规范来描述服务，一个或多个来自于同一个Docker镜像的容器可被定义为一个基本服务。在同一个应用栈里，当一个服务（消费者）被链接到另外一个服务（生产者）时，会自动产生一条DNS记录，来匹配每个容器的实例。该记录使得服务（生产者）能够被访问到。在Rancher中创建服务的其它益处包括：

- 服务高可用 (HA) - 可以使Rancher来自动监控容器的状态，从而确保服务内的启动的容器维持在预定的数量。
- 健康检查 - 设定阈值，来监控容器的健康状况。
- 添加负载均衡器 - 为服务添加一个基本的负载均衡器，该负载均衡器使用HAProxy。
- 添加外部服务 - 可以添加任意的IP地址做为可以使用和访问到的服务。
- 添加服务别名 - 可以添加一条DNS记录，使得服务可以被访问到。

更多信息，请参考 添加服务, 添加负载均衡器, 添加外部服务 或者 添加服务别名.

负载均衡器

Rancher使用HAProxy来构建一个被管理的负载均衡器，该负载均衡可以被定义部署到多台主机上。一个负载均衡器可用来将网络和应用流量分配到多个单个容器里，只需要将这些容器添加到与负载均衡器相链接的服务里。这个链接到负载均衡器的服务，可以由Rancher自动注册为负载均衡的目标服务。

分布式DNS服务

Rancher本身内置具备高可用控制引擎的轻量级DNS服务，利用该服务，Rancher构建了一个分布式DNS服务。每个健康的容器被链接到一个服务或者被添加到一个服务别名的时候，该容器会被自动添加到DNS记录里的该服务中。当查询该服务名时，DNS服务会返回该服务中所有容器的随机IP列表。

- 缺省情况下, 在同一应用栈里的所有服务不需要显式的链接定义，就会被添加到DNS服务中。
  - 你可以通过服务名称来解析同一应用栈的容器地址。
  - 如果你的服务需要一个自定义的DNS名称，该名称与原服务名称不同，你将需要一个链接来得到这个自定义的DNS名称。
  - 对负载均衡来说，仍旧需要链接定义才能访问到目标服务。
  - 如果定义了服务别名，链接就仍旧时必须定义的。
- 如果想使得在不同应用栈的服务可以被解析，你需要显式地定义链接。

因为Rancher的覆盖（overlay）网络为每个容器定义了一个明确的IP地址，你不需要再操心处理端口映射，也不需要处理类似同一服务下的容器使用不同的端口这样的复杂情况。这样，利用一个基本的DNS服务就足可以实现服务发现。

健康检查

Rancher构建了一个完善的容器健康状况监测系统。该系统通过在多个主机上运行的网络代理，来实施对容器和服务的分布式健康状况检查。这些网络代理使用内置的HAProxy来验证应用的健康状态。当一个容器或一个服务定义了健康检查时，每个容器就会被最多三个网络代理程序来监控，这些网络代理程序运行在不同于该容器父节点的其它三个物理节点上。只有当至少一个HAProxy的实例表明健康检查“通过”时，该容器才会被认为是健康的。

> 备注: 一个例外情况是，当你的环境里只包含了一个单一的主机时，健康检查就会被同一个主机节点上的网络代理来执行。

Rancher的去中心化的健康检查模式比客户端模式要有效得多，另外，使用HAProxy来进行健康检查，那些原先只能在负载均衡器定义的应用级别负载均衡策略，都可以在服务里进行定义，从而实现在各层次上去监控应用和服务的健康状况。

更多信息，例如健康检查失败的不同场景以及Rancher如何显示服务，请参考 健康检查. 你也可以获取更多信息关于如何通过使用 rancher-compose 或 在UI里来设置健康检查.

服务高可用

Rancher会持续地监控服务里容器的状态，并且主动地进行管理，确保服务保持在期望的规模。当健康的容器数量比服务里原先定义的数量少（或者甚至多）的情况下，系统会自动触发动作，启动（或关闭）容器。 通常这种情况是由于一个节点不可用，一个容器失败或者容

器／服务没有通过健康检查。

服务升级

Rancher支持服务升级的概念，该功能是通过允许用户为某个给定的服务来使用负载均衡或者应用服务别名来实现的。这两种方式，都可以为需要该服务的现有负载关键一个静态的目标。一旦该设定建立，某个底层的服务可以被Clone为一个新服务，通过隔离的测试来进行校验，当就绪时，通过负载均衡或者定义服务别名的方式加入到应用栈。 而原有的服务当废弃时可以在系统中删除。最终，所有的网络或应用流量都将被分流到新的服务。

Rancher Compose

Rancher实施和发布了一个命令行工具，该工具称为“rancher-compose”，与docker-compose的命名类似. 该工具可以引用同样的docker-compose.yml模版将应用栈在Rancher中部署。Rancher-compose的工具还支持rancher-compose.yml文件，该文件扩展了docker-compose.yml的定义，允许定义更多的属性，如容器规模, 负载均衡策略，健康检查策略以及外部链接等，这些定义目前在docker-compose里还不能定义.

更多信息, 请参见 rancher-compose.

应用栈（Stacks）

Rancher应用栈与docker－compose所定义的项目概念类似。 它代表着一组服务，来构成一个典型的应用或工作负载。

容器调度

Rancher支持容器调度策略，采用与Docker Swarm兼容的模型。 包括如下策略：

- 端口冲突
- 共享存储卷
- 主机标签
- 共享网络栈: --net=container:dependency
- 严格的和软性定义的亲和／反亲和规则，这些规则通过使用env变量(Swarm)和label(Rancher)来定义。

另外， Rancher支持调度服务触发器，允许用户定义特定的规则，如“添加主机”或“主机标签”，从而实现自动地将服务扩展到具有特定标签到主机上。

关于如何调度容器到更多信息，请参考如何使用UI的labels 和 调度规则 或者如何使用labels in rancher-compose.

Sidekicks

Rancher允许用户通过定义Sidekicks来定义一组容器，这组容器和主容器总是运行在同一主机节点上、同时被调度以及同时被伸缩。 一个服务拥有一个或多个sidekick容器，一个典型的例子是用来在多个容器里支持存储卷(即 `--volumes_from`)和网络(即 `--net=container`)的定义.

更多信息, 请参考sidekicks with rancher-compose.

Rancher 基础服务

健康检查

Rancher 通过在它的主机上运行托管网络代理端实现了健康监控系统用来协调容器和服务的分布式健康检查。这些网络代理端在内部使用 HAProxy 来验证你的应用的健康状态。当无论单独容器还是服务的健康检查被开启后，每个容器都会被多至三个网络代理端所监控，代理端运行于不同于容器主机的主机上。如果至少有一个 HAProxy 实例汇报“通过”的健康检查，容器就会被认为是健康的，当所有 HAProxy 实例都汇报“不健康”的健康检查，就认为容器是不健康的。

> 注意: 此模式仅有的例外情况是：当你的环境中仅包含一个单主机，在这种情况下健康检查将只会在这台主机上执行。

Rancher 处理网络区分并且那比基于客户端的健康检查更有效率。通过使用 HAProxy 来执行健康检查，Rancher 允许用户跨越应用和负载均衡来指定相同的健康检查策略。

> 注意: 监控检查仅对使用托管网络的服务有效。如果你选择了任何其他的网络选项，它将不会被监控。

配置

使用如下选项来配置健康检查：

Check type: 这里有两种检查 - *TCP Connection Opens* (仅确认端口打开) 和 *HTTP Responds 2xx/3xx* (执行一个 HTTP 请求并确保接受了正确的响应)。

HTTP request: 如果检查类型是 *HTTP Responds 2xx/3xx*，你必须指定查询的 URL 路径。你能选择请求方法 (`GET` , `POST` , etc) 和 HTTP 版本 (`HTTP/1.0` , `HTTP/1.1`)。

Port: 执行检查的端口。

Check interval: 检查执行的毫秒数间隔。

Timeout: 在一次无回应的检查超时前的毫秒数。

Healthy threshold: 在容器(当前标记为非健康)被再次认为是健康前成功的检查响应次数。

Unhealthy threshold: 在容器(当前标记为健康)被认为是非健康前失败的检查响应次数。

失败场景

场景	响应
被监控容器停止响应健康检查	所有激活监控此容器的 HAProxy 实例将会侦测到失败并标记此容器为“非健康”。如果这个容器是一个服务的一部分，Rancher 将会通过它的 HA 功能恢复此服务到它预定义的数量。
运行带有健康检查容器的主机丢失了网络连接或主机上的代理端。	当主机的网络连接丢失，从 Rancher 服务端到代理端的连接也会丢失。当代理端不可访问，主机会被标记为"reconnecting"。 此时所已知的是 Rancher 服务端无法连上那台主机的代理端。Rancher 中的健康检查针对的是容器本身而不是主机；结果是，对所有激活的 HAProxy 实例，容器都不可达。如果这个容器是一个服务的一部分，Rancher 将会通过他的 HA 功能恢复此服务到它预定义的数量。
运行带有健康检查容器的主机完全失效。	当主机遭受完全失效例如电源中断，从 Rancher 服务端到代理端的连接中断。 当代理端不可访问，主机会被标记为"reconnecting"。此时所已知的是 Rancher 服务端无法连上那台主机的代理端。Rancher 中的健康检查针对的是容器本身而不是主机；结果是，对所有激活的 HAProxy 实例，容器都不可达。如果这个容器是一个服务的一部分，Rancher 将会通过他的 HA 功能恢复此服务到它预定义的数量。
一台主机的代理端失效但主机保持在线，并且容器继续运行并能通过健康检查。	在此实例中，如同先前的案例，从 Rancher 服务端到代理端的连接中断。当代理端不可访问，主机会被标记为"reconnecting"。此时所已知的是 Rancher 服务端无法连上那台主机的代理端。Rancher 中的健康检查针对的是容器本身而不是主机；结果是，对所有激活的 HAProxy 实例，容器都不可达。如果这个容器是一个服务的一部分，Rancher 将会通过他的 HA 功能恢复此服务到它预定义的数量。勘误：*by alfredhuang211*，此例子中，容器应该还是可达的，但是*rancher*英文原文仍然写的是不可达，猜测不知是撰写错误还是本人理解有限。因为容器可达，健康检查仍然可以通过，认为容器仍然健康。

Rancher 内部 DNS 服务

在 Rancher 里，我们有自己的内部 DNS 服务，允许同一个 cattle 环境 里的服务能解析到另外任意一个服务。

环境里的所有服务都通过 `<service_name>` 被解析，并且不需要在服务间建立连接。对于处于不同栈的其他任何服务，你需要通过 `<service_name>.<stack_name>` 而不是 `<service_name>` 。如果你想要使用不同名字来解析服务，你能设置一个连接，以便服务能够通过服务别名解析。

通过连接设置服务别名

在 UI 里，当 添加服务 时, 展开 **Service Links** 部分，选择服务，提供别名。

如果你是使用 `rancher-compose` 来 添加服务， `docker-compose.yml` 可以使用 `links` 或 `external_links` 指令。

```
service1:
  image: wordpress
  # 如果另外的服务是在同一个应用堆栈中
  links:
    # <service_name>:<service_alias>
    - service2:mysql
  # 如果另外的服务是在不同的应用堆栈中
  external_links:
    # <service_name>:<service_alias>
    - service3:mysql
```

从容器和链接

当启动一个服务，你可能需要服务一起一直启动在同一主机上。特殊用例是当试图使用另一个服务的 `volumes_from` 或 `net` 。当无论是通过UI 还是使用 rancher-compose创建一个伙伴关系，服务都自动通过他们的名字互相解析到。我们当前不支持在伙伴服务内通过链接/外部链接创建服务别名。

当创建伙伴关系，总是有一个主服务和若干伙伴服务。合在一起，他们被认为是一个单独的运行配置。运行配置会作为一组容器被部署在一台主机上，主服务一个，另外每个伙伴服务一个。在运行配置内的任何一个服务，你都能通过他们的名称解析到主服务或者伙伴服务。对于运行配置外的任何服务，主服务能够通过名字解析到，但伙伴服务仅能通过 `<sidekick_name>.<primary_service_name>` 解析到。

容器名

所有的容器都能通过他们的名字被全局解析到，因为每个服务的容器名在每个环境内都是独一无二的。不需要在后面加上服务名或栈名。

例子

Ping 同一栈里的服务

如果你 exec 进入容器的 shell，你能通过服务名 ping 同一栈内的其他服务。

在我们的例子中，栈名为 `stackA` ，带有两个服务 `foo` 和 `bar` 。

在 exec 进入 `foo` 服务的一个容器中后，你能 ping `bar` 服务。

```
$ ping bar
PING bar.stacka.rancher.internal (10.42.x.x) 58(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.63 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.13 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.07 ms
```

Ping 不同栈里的服务

对于不同栈内的服务，你能通过使用 `<service_name>.<stack_name>` ping 不同栈内的服务。

在我们的例子中，我们有一个栈为 `stackA` ，带有服务 `foo` ，然后我们还有一个栈为 `stackB` ，带有服务 `bar` 。

如果我们 exec 进入服务 `foo` 的一个容器中，你能通过 `foo.stackB` ping `bar` 服务。

```
$ ping bar.stackb
PING bar.stackb (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.43 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.15 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.27 ms
```

Ping 伙伴服务

取决于你从哪个服务发起 ping，你能通过 `<sidekick_name>` 或 `<sidekick_name>.<primary_service_name>` 到达伙伴服务。

在我们的例子中，我们有一个栈为 `stackA` ，包含服务 `foo` ，带有伙伴服务 `bar`,还有个服务为 `hello` 。我们还有个栈为 `stackB` ，包含服务 `world` 。

如果我们 exec 进入 `foo` 服务的一个容器，你能通过他的名字直接 ping `bar` 服务。

```
# Inside  one of the containers in the `foo` service, which `bar` is a sidekick to.
$ ping bar
PING bar.foo.stacka.rancher.internal (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=64 time=0.111 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=64 time=0.114 ms
```

如果我们 exec 进入 `hello` 服务的一个容器，在同一个栈内，你能通过 `foo` ping `foo` 服务，通过 `bar.foo` ping `bar` 伙伴服务。

```
# 在`hello` 服务中的一个容器内部，它不属于 service/sidekick 服务的一部分
# Ping the primary service (i.e. foo)
$ ping foo
PING foo.stacka.rancher.internal (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.04 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.40 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.07 ms
# Ping the sidekick service (i.e. bar)
$ ping bar.foo
PING bar.foo (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.01 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.12 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.05 ms
```

如果我们 exec 进入 `world` 服务的一个容器，在不同栈内，你能通过 `foo.stacka` ping `foo` 服务，通过 `bar.foo.stacka` ping `bar` 伙伴服务。

```
# Inside one of the containers in the `world` service, which is in a different stack
# Ping the primary service (i.e. foo)
$ ping foo.stacka
PING foo.stacka (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.13 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.05 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.29 ms
# Ping the sidekick service (i.e. bar)
$ ping bar.foo.stacka
PING bar.foo.stacka (10.42.x.x) 56(84) bytes of data.
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.23 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.00 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=0.994 ms
```

Pinging 容器名

从任何容器，你能通过他们的名字 ping 环境内的其他容器，而不用管他们是否在不同栈或服务内。

在我们的例子中，我们有一个栈为 `stackA` ，包含服务 `foo` 。我们还有个栈为 `stackB` ，包含服务 `bar` 。 容器的名字是 `<environment_name>_<service_name>_<number>` 。

如果我们 exec 进入 `foo` 服务的一个容器，你能 ping `bar` 服务内的容器。

```
$ ping stackB_bar_1
PING stackB_bar_1.rancher.internal (10.42.x.x): 56 data bytes
64 bytes from 10.42.x.x: icmp_seq=1 ttl=62 time=1.994 ms
64 bytes from 10.42.x.x: icmp_seq=2 ttl=62 time=1.090 ms
64 bytes from 10.42.x.x: icmp_seq=3 ttl=62 time=1.100 ms
```

使用 Route 53 做外部 DNS

作为 Rancher catalog 的一部分，Rancher 提供了与Amazon Route53 DNS 集成的 DNS 服务。当启动这个服务，有一个 route53 的容器被启动。这个容器会监听Rancer 元数据服务器和事件，会基于元数据的变化更新 DNS 记录，并更新到 Route53 服务中。

最佳实践

- 在您部署的 Rancher 的每个环境中，应该有一个容器数量为1的 `route53` 服务。
- 多套 Rancher 部署环境应该不能共享相同的 `hosted zone` 。

启动 Route53 服务

点击 **Catalog** 标签，您能选中**Route53 DNS Stack**。

输入一个 名字，输入关于这个堆栈的 描述。

在**Configuration Options**中，你需要提供一下信息：

名称	值
AWS Access Key	您账户AWS API的 Access key
AWS Secret Key	您账户AWS API的 Secret key
AWS Region	AWS的 Region 名。我们建议选择离您所地最近的区。更新的信息将会被发送到该 AWS API 端点，Rancher Route53 会随时及时更新 DNS 更新请求。
Hosted Zone	Route53 的 hosted zone。这个 zone 需要提前创建好。

在填写了这些字段后，点击 **Create**按钮。堆栈将被创建出来带有 `route53` 的服务，而您只需要做启动这个服务即可。

使用 Route53 服务

这个 `route53` 服务只会在服务有端口暴露在主机上时，来未知生成 DNS 记录。对于每一个 Rancher 生成的 DNS 记录，它会按下面的形式创建出 fqdn：

```
fqdn=<serviceName>.<stackName>.<environmentName>.<yourHostedZoneName>
```

在 AWS 的 Route 53上，把以 Record Set 的形式被表达：name=fqdn 和 value=[服务被部署的主机的 IP 地址。 Rancher `route53` 服务将只管理以 . 结尾的 Record Set，默认的 300 秒。

一旦在 AWS 的 Route 53里配置了 DNS 记录，被创建的 fqdn 记录将被传回 Rancher，会被配置在 **service.fqdn** 字段。您能通过点击服务的 **View in API**下拉菜单后，搜索找到 **fqdn**找到对应的 fqdn 记录。

当在浏览器中使用 fqdn 时，它会被指向服务中的一个容器中。如果一个服务中容器的 IP 地址会有任何变化，这些变化将也被更新到 AWS Route 53服务中。而从用户使用 fqdn 域名访问服务的角度看，则不会体验到有任何变化。

> 注意: 在 `route53` 服务启动了以后，对于已经部署了的有主机端口的服务将也会得到一个 fqdn。

删除 **Route53** 服务

当从 Rancher 中删除了 `route53` 服务之后，Amazon Route 53中的 Record Set 将 不会 被自动删除。它们还是需要登录了您的 Amazon AWS 账号之后再手工删除掉。

为外部 **DNS** 使用特定的 **IP**

默认，Rancher DNS 所使用的主机IP，是注册到 Rancher 中是所指定的IP。这里会有这样一种用例，被配置在 Rancher 中的主机使用私有网络通信，而这个主机将需要通过外部 DNS 使用公网 IP 来对外暴露服务。在这种情况下在你需要制定外部 DNS 所使用的IP，您需要在使用外部 DNS 服务前为主机添加 主机标签 。

在你启动外部 DNS 服务之前，请添加下面的主机标签。标签的值是 Rancher 的 Route 53 服务会用到的。如果没有设置这个标签的话，Rancher 的 Route53 DNS 服务器就会直接使用主机 IP。

```
io.rancher.host.external_dns_ip=<IP_TO_BE_USED_FOR_EXTERNAL_DNS>
```

元数据服务

Martin 初始化版本翻译中，欢迎随后 review

使用 Rancher 的元数据服务，你能在任意 Rancher 所管理的何容器内部来查询到关于容器所管理的网络和的相关信息。 元数据可以涉及到容器自身、所在服务或者堆栈，以及容器所允许的主机，等等。元数据默认格式是 JSON 。

容器可以通过以下几种方式在 Rancher 管理的网络上启动。

- 通过图形界面 UI, 一个服务/容器 被启动时使用了 *Managed* 网络参数。默认情况下，服务网络已经被设置为 *Managed* 。
- 通过 Rancher-Compose 命令行，一个服务/容器，当没有被设置为网络模式 (`net`)，在 Rancher 管理的网络上被启动。
- 当用 docker 原生命令 启动容器, 如果你在 `docker run` 命令中加入标签 `io.rancher.container.network=true` ，这样容器就加入了 Rancher 的管理网络。

> 注意: 元数据服务对于Rancher 的系统容器是不可用的，如：Network Agent 和 LB Agent 容器。

如何获取到元数据

在 Rancher 的图形界面上，您能从容器的下拉菜单 **Execute Shell** 上进入到容器的命令行。下拉菜单在鼠标滑过容器的时候可以出现。

你可以使用 curl 命令来过去元数据。

```
# 如果 curl 没有安装，安装它
$ apt-get install curl
# curl 命令可以返回纯文本的相应结果
$ curl http://rancher-metadata/<version>/<path>
```

访问路径依赖于你想访问什么样元数据和想得到的相应结果的格式。

元数据	路径	描述
容器	`self/container`	提供的元数据是您执行元数据查询命令的容器自身的信息
容器所在的服务	`self/service`	提供的元数据是您执行元数据查询命令的容器所属服务的信息
容器所在的堆栈	`self/stack`	提供的元数据是您执行元数据查询命令的容器所属堆栈的信息
容器所运行的主机	`self/host`	提供的元数据是您执行元数据查询命令的容器所运行于主机的信息
其它容器	`containers`	提供所有其它容器的元数据。用纯文本格式，它可返回了所有容器索引后的清单。用 JSON 格式，它可返回所有容器的所有元数据信息。使用索引或者容器名称在路径中，你可以获取特定容器的元数据。
其它服务	`services`	提供所有其它服务的元数据。用纯文本格式，能返回所有服务的索引编号。用 JSON 格式，它能返回所有服务的元数据信息。在路径中使用索引编号或者名称，可以获得一个特定服务的元数据信息。如果去查看容器相关信息，在V1（`2015-07-25`）中只能返回容器的名称，而在V2（`2015-12-19`）中则可以容器对象清单。
其它堆栈	`stacks/<stack-name>`	提供所有其它堆栈的元数据。用纯文本格式，能返回所有堆栈的索引编号。用 JSON 格式，它能返回所有堆栈的元数据信息。在路径中使用索引编号或者名称，可以获得一个特定堆栈的元数据信息。如果去查看容器相关信息，在V1（`2015-07-25`）中只能返回容器的名称，而在V2（`2015-12-19`）中则可以容器对象清单。

元数据的版本

在 `curl` 命令里，我们强烈建议使用一个的定的版本，而不总是使用 `latest` 。

> 注意: 我们对 `latest` 版本会持续更改，不同版本返回的数据可能不同，因此可能会影响到您的代码的兼容性。

元数据服务的版本是基于日期的。

版本参考	版本
V2	2015-12-19
V1	2015-07-25

版本的差异

V1 vs. V2

当使用路径 `/services/<service-name>/containers` 或 `</stacks/<stack-name>/services/<service-name>/containers` 查看容器元数据信息时，V1 返回容器的名称，而 V2 返回所有容器对象。元数据服务的 V2 版本可以提供更多信息。

实例

在 Rancher 中，有一个堆栈名为 `foostack` ，它包含一个由三个容器组成的名为 `barservice` 的服务。

```
# 使用 V1 返回的只是服务的容器名称
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-07-25/service
s/barservice/containers'
["foostack_barservice_1", "foostack_barservice_2", "foostack_barservice_1"]

# 使用 V2 返回的是服务中所有容器对象
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/service
s/barservice/containers'
[{"create_index":1, "health_state":null,"host_uuid":...
...
# 服务里所有容器的元数据清单
...
...}]

# 使用 V2, 你能详细查看某个特定容器对象
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/service
s/barservice/containers/foostack_barservice_1'
[{"create_index":1, "health_state":null,"host_uuid":...
...
# 服务里所有容器的详细信息清单
...
...}]

# 使用 /stacks/<服务-名称>, 你能详细查看服务和容器

# 使用 V1 只能返回服务中容器的名称
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-07-25/stacks/
foostack/services/barservice/containers'
["foostack_barservice_1", "foostack_barservice_2", "foostack_barservice_1"]

# 使用 V2 能返回服务中所有容器对象详情清单
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/stacks/
foostack/services/barservice/containers'
[{"create_index":1, "health_state":null,"host_uuid":...
...
# 服务里所有容器的详细信息清单
...
...}]
```

纯文本 vs JSON 格式

元数据信息能返回成纯文本合作和 JSON 格式。基于你怎样使用元数据的需求，你能选择任意格式。

纯文本

在执行 curl 命令时，您会从所请求的路径中获得纯文本。你可以从路径的最顶级开始查询，根据返回的可用的键值信息继续向下层访问，直到元数据服务返回了您想查询的数据。

```
$ curl 'http://rancher-metadata/2015-12-19/self/container'
create_index
health_state
host_uuid
hostname
ips/
labels/
name
ports/
primary_ip
service_name
stack_name
start_count
uuid
$ curl 'http://rancher-metadata/2015-12-19/self/container/name'
# 注意: Curl 不能换行，因此最后一行的返回值和命令提示符出现在同一行行。
Default_Example_1$root@<container_id>
$ curl 'http://rancher-metadata/2015-12-19/self/container/label/io.rancher.stack.name'
Default$root@<container_id>
# 数组可以通过索引号和名称来返回值
$ curl 'http://rancher-metadata/2015-12-19/services'
0=Example
# 你可以在路径中使用索引号或者名称
$ curl 'http://rancher-metadata/2015-12-19/services/0'
$ curl 'http://rancher-metadata/2015-12-19/services/Example'
```

JSON

把 `Accept: application/json` 添加在 curl 命令的请求标记中，可以返回 JSON 格式的数据。

```
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/self/container'
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/self/stack'
# 返回某个堆栈总其它服务的元数据
$ curl --header 'Accept: application/json' 'http://rancher-metadata/2015-12-19/services/<service-name>'
```

元数据字段

容器

字段	描述
`create_index`	在服务里容器被启动的顺序编号，例如：2意味着它应该是在服务中第二个被启动的。注意：Crete_index 是不会被重复使用的。如果您的服务有两个容器，当你删除了第二个容器后，下一个被启动的容器得到的 create_index 编号是3，尽管这个服务中总共只有2个容器。
`health_state`	如果 健康检查 被启用用了，这是容器健康状态信息。
`host_uuid`	Rancher 服务器分配给主机的唯一识别标识。
`hostname`	容器的主机名。
`ips`	当多网卡被支持了，它将显示 IP 地址清单。
`labels`	容器上的标签 清单。标签的格式是 `key : value` 。
`name`	容器的名称
`ports`	容器内使用的端口 清单. 端口的格式 `hostIP:publicIP:privateIP[/protocol]` 。
`primary_ip`	容器 IP 地址
`service_index`	容器所在的服务的编号
`service_name`	服务的名字 (如果存在)
`stack_name`	服务所在的堆栈的名字 (如果存在)
`start_count`	容器被自动的次数。
`uuid`	Rancher 分配给容器的唯一标识。

服务

字段	描述
containers	服务中容器的名称清单。
create_index	Create_index of the last container created of the service. Note: Create_index is never reused. If you had a service with 2 containers and deleted the 2nd container, the create_index will be 2. The next container that gets launched for the service would update the create_index to 3 even though there are only 2 containers.
expose	
external_ips	外部服务 的外部 IP 地址清单
fqdn	服务的 FQDN 全域名
hostname	外部服务 的CNAME
kind	Rancher 服务的类型
labels	服务上的标签 清单。标签的格式是 key:value .
links	所连接的服务。 连接的格式是 stack_name/service_name:service_alias 。 links 会显示所有的键值(例如： stack_name/service_name 所有连接) 和返回 service_alias , 你会需要继续向下查看来找到特定的键值。
metadata	用户添加的元数据
name	服务的名称
ports	在服务中所使用的端口. 端口的格式是 hostIP:publicIP:privateIP[/protocol] 。
scale	服务的数量
sidekicks	sidekicks 的服务名清单
stack_name	服务所在的堆栈的名称
uuid	Rancher 分配给服务的唯一标识

堆栈

字段	描述
environment_name	堆栈所在 环境 的名称
name	堆栈 的名称
services	堆栈中的服务的清单
uuid	Rancher 分配给堆栈的唯一标识。

主机

字段	描述
`agent_ip`	Rancher Agent 的 IP 地址，例如： `CATTLE_AGENT_IP` 环境变量的值。
`hostId`	在某个环境中主机的唯一标识
`labels`	主机标签 清单。标签格式是 `key:value`
`name`	主机 名称
`uuid`	Rancher 分配给主机的唯一标识

给服务添加用户元数据

Rancher 可以让用户给服务增加自己的元数据。目前，这个功能只能通过 rancher-compose 来实现，并且元数据需要是 `rancher-compose.yml` 文件的一部分。在 `metadata` 键里，yaml 将被解析为 JSON 格式用于元数据服务。

`rancher-compose.yml` 示例

```
service:
  # Scale of service
  scale: 3
  # User added metadata
  metadata:
    example:
      name: hello
      value: world
    example2:
      foo: bar
```

在这个服务启动之后，你可以通过元数据服务找到这个信息在 `.../self/service/metadata` 或者 `.../services/<service_id>/metadata` 中。

JSON 查询

```
$ curl --header 'Accept: application/json' 'http://rancher-metadata/latest/self/servic
e/metadata'
{"example":{"name":"hello","value":"world"},"example2":{"foo":"bar"}}
```

Plaintext 查询

```
$ curl 'http://rancher-metadata/latest/self/service/metadata'
example/
$ curl 'http://rancher-metadata/latest/self/service/metadata/example'
name
value
$ curl 'http://rancher-metadata/latest/self/service/metadata/example/name'
# Note: Curl will not provide a new line, so single values will be on same line as the
 command prompt
hello$root@<container_id>
```

存储服务

Alan已领取，翻译中。

Storage Service

In the Rancher catalog, Rancher provides storage services that are capable of exposing volumes to containers. Currently, the storage services are pre-fixed with Convoy in the catalog.

Prerequisites •You have the underlying storage system properly deployed and accessible to Rancher services

Limitations •Only one Convoy stack can be deployed once per one Rancher environment. •Users are able to create service/containers and models exactly the same behavior as if you deployed a container using native Docker 1.10.3 commands. When creating a new container, the following rules apply: ◦If the volume name (e.g. foo:/path/in/container) is specified with a driver name (e.g. Convoy-Gluster), the container will be deployed on one of the hosts that belong to the Convoy-Gluster Storage Pool with a new volume "foo" created. ◦If the volume name (e.g. foo:/path/in/container) is specified with or without a driver name and "foo" exists in Rancher, the container will be launched on a host from that Storage Pool that has access to "foo". ◦If the volume name (e.g. foo:/path/in/container) is specified with NO driver name and "foo" DOES NOT exist in Rancher, it will be created as a local disk volume for that container. At this point, normal scheduling rules apply.

Setting up the Storage Service

From the Catalog tab, select one of the Convoy services.

By default, we've auto-populated the required fields. Edit the fields to your desired choices. After filling in the form, click on Create.

Viewing Storage Pools

After your storage service has been launched, a storage pool has been created and is viewable in Infrastructure -> Storage Pools. You will be able to see all the Convoy storage services that are running in your environment. The name of the storage pool is derived from the name of the stack.

In each storage pool, the hosts that have the storage service running will be listed. Besides the list of hosts, the lists of volumes used in the storage pool is also listed. For each volume, you can see the name of the volume (i.e. the name of the volume on the host(s)), and the mounts of each volume. For each mount, there is the container name and the directory path inside the container.

Using the Storage Service in the UI

After your storage service has been launched, a storage pool has been created and is viewable in Infrastructure -> Storage Pools. Services can start using the shared storage. In the Volumes tab, provide a volume and a volume driver. The volume will be in the same syntax as Docker, :</path/in/container> . Docker volumes default to mount in read-write mode, but you can set it to be mounted read-only by adding the :ro at the end of the volume. The volume driver will be the name of the storage pool, that was created after launching the Convoy storage service.

Using the Storage Service with Rancher-Compose

After the storage service has been launched, you can start using the storage service as a volume_driver in the docker-compose.yml . The volume_driver would be the name of the storage pool.

test: tty: true image: ubuntu:14.04.3 stdin_open: true volumes:

- volume_name_on_host:/path/in/container volume_driver:

Adding Volumes to Storage Pools

A volume can be added to a storage pool with two methods: 1.Upon starting a service that has a volume and a volume driver, that is the storage service. When the service is started, the volume is created on all hosts and mounted in all containers. The volume is also added to the storage pool in Rancher. 2.In Infrastructure -> Storage Pools, click on Add Volume. Provide the name of the volume for the specified storage pool.

Example using GlusterFS

In this example, we're going to provide an example of how to use GlusterFS to have a shared storage across hosts. 1.Launch Gluster FS ◦In the Catalog tab, click on View Details of the Gluster FS service. ◦If desired, edit the Name of the stack for Gluster FS and add a description. ◦In the Configuration Options, edit the volume name. By default, Rancher has given the volume name my_vol . ◦Click on Launch to start the Gluster FS service. This will take a minute or two. You will be able to view the services in Applications -> Stacks.

2.Launch Convoy Gluster ◦In the Catalog tab, click on View Details of the Convoy Gluster service.

◦If desired, edit the Name of the stack for Convoy Gluster and add a description. ◦In the Configuration Options, edit the Volume Name, if you have changed the volume name while launching the Gluster FS service. ◦Select the Gluster FS service that is running in Rancher. The service is named glusterfs-server . ◦Click on Launch to start the Convoy Gluster service. The convoy-gluster service will be deployed on every host. You will be able to view the services in Applications -> System. ◦Check that the storage pool (convoy-gluster) has been created in the Infrastructure -> Storage Pools tab. Note: The name of the storage pool is derived from the name of the stack.

3.Launch Service using the Convoy Gluster service ◦Click on Add Service in a different stack. Set up your service as you typically would. ■For this example, I'll be using the ubuntu:14.04.3 image to provide how the storage service is working and a scale of 10 .

◦In the Volumes tab, add the name of your volume. The naming convention of the volume will be the same as Docker, :</path/in/container> . Docker volumes default to mount in read-write mode, but you can set it to be mounted read-only by adding the :ro at the end of the volume. ■For this example, I have used glustervol1:/testvolume as the volume name.

◦In the Volumes tab, the Volume Driver will be the name of the storage pool that was created. ◦Click on Create to create your service. After the service is created, start the service.

4.Checking that the Services across Hosts are sharing volumes ◦In the detailed page of your service, you can see all the containers launched on different hosts. ◦Select a container on one of the hosts and use the container's dropdown to select Execute Shell. In the container, check that the directory that you had set in the service creation exists. Create a file in the shared volume directly.

$ cd testvolume $ vi test.yml $ ls test.yml ◦Select a container on a different host and use the container's dropdown to select Execute Shell. In the container, check that the directory and file that was created in the previous container exists.

$ cd testvolume $ ls test.yml

5.Checking the Volume Driver on the Host ◦Use docker volume ls to view the list of Docker volumes. One of the entries will be using the convoy_gluster driver with the volume name that was created in the service. In my example, glustervol1 will be listed.

审计日志

只有管理者才能存取审计日志。 你可以在 Admin -> Audit Log 找到审计日志。

Rancher 的审计日志是集合了不同类型的事件。

- 任何前綴是 api 调用我们的 API 时。 这个事件将会记录谁执行这个动作和怎么詷用这个 API (例如：透过用户界面、透过 API key)
- 其他的有让 Rancher 服务端动作而没有前綴的 api 事件。例如实例在一个服务的容器在处理过程，会记录在 `instance.create` 事件中。

系统配置

本文档首稿正在编辑中，请随后帮忙 review。

访问控制

Access Control

What is Access Control?

The first account that authenticates with Rancher will become the **admin** of the account. For more information, see permissions of an admin.

Enabling Access Control

In the **Admin** tab, click **Access Control**.

Currently, Rancher supports three methods of authentication: Active Directory, GitHub, Local and OpenLDAP. After authenticating your Rancher instance, Access Control will be considered enabled. With Access Control enabled, you will be able to manage different environments and share them with different groups of people.

When Access Control is enabled, the API is locked and requires either being authenticated as a user or by using an API key to access it.

Active Directory

Select the **Active Directory** icon. If you want to use Active Directory using TLS, ensure that you have started Rancher server with the appropriate certificate. Fill in the sections and authenticate Rancher by clicking **Authenticate**. When Active Directory is enabled, you'll automatically be logged in as the username that was authenticated. You will also be logged in as an admin of Rancher.

GitHub

Select the **GitHub** icon and follow the directions in the UI to register Rancher as a GitHub application. After clicking **Authenticate with GitHub**, Access Control is enabled and you are automatically logged into Rancher with your GitHub login credentials and as an admin of Rancher.

Local Authentication

Local authentication allows you to create your own set of accounts that is saved in the Rancher database.

Select the **Local** icon. Create an admin user by providing the **Login Username**, **Full Name**, and **Password**. Click **Enable Local Auth** to turn on local authentication. By clicking this button, the admin is created and saved in the database. You are automatically logged into the Rancher instance as the admin account that was just created.

OpenLDAP

Select the **OpenLDAP** icon. If you want to use a OpenLDAP using TLS, ensure that you have started Rancher server with the appropriate certificate. Fill in the sections and authenticate Rancher by clicking **Authenticate**. When OpenLDAP is enabled, you'll automatically be logged in as the username that was authenticated. You will also be logged in as an admin of Rancher.

Site Access

Anyone who is a member or owner of an environment will also have access to the Rancher instance. Alternatively, you can invite members to access the Rancher instance, but when they log in, they will have their own environment. These members, who are added by using **Access Control** or **Accounts** options, will not be able to see other environments until they are added as a member to them.

Anyone with the permissions to view the Rancher instance will be given user permissions. However, they will not be able to view the **Admin** tab. In order for users to view different environments, they will need to be added by an owner to the environment.

Active Directory/OpenLDAP

Once Active Directory or OpenLDAP is enabled, any user will be able to access the Rancher site. Eventually, we will be adding in the ability to restrict the Rancher instance to specific users and group members. Rancher still has environments to keep resources protected from other users and groups, but anyone part of your LDAP server will be able to access the Rancher instance.

GitHub

If you wanted to add users to Rancher without sharing your environment, you can add them in the **Site Access** section. Click the **Customize** button.

Option 1: Allow any GitHub user

This would allow anyone with a GitHub account to access your site.

Option 2: Restrict to specific GitHub users and organization members

By clicking the drop-down menu icon next to the **+** button, you'll see the list of organizations that you are a member of. When you click any of those organizations, Rancher adds their name to the list of *Authorized Users and Organizations*. You can also add individual users by typing in their username into the text box and clicking the **+**. This will just add the users/organizations to the list, but you will still need to save!

> **Note: Save your configuration!** For either option, you must click the **Save authorization configuration** button in order for any changes to take affect. If you leave the page before saving, your changes will be lost.

Local Authentication

Once local authentication is enabled, the admin can create additional admins/users by accessing the **Admin** > **Accounts** tab. Click **Add Account** and fill in the details of the account you want to add. You can select their account type as an **Admin** or **User**. An admin has the ability to view the **Admin** tab while users of Rancher instance would not have the visibility to the tab.

Once an account has been created, the admin/user can be added to any environments.

Account Types

The account type determines whether or not an account will have access to the admin tab. For each environment in Rancher, there are membership roles that provide different level of access for a specific environment.

Admin

The first user that authenticates Rancher becomes an admin of Rancher. Only admins will have permissions to view the **Admin** tab. This includes the ability to configure **Access Control**, **Accounts**, **Settings**, and **Processes**.

When managing environment, admins have the ability to view all the environments in Rancher even if the admin is not added as a member to the environment. In an admin's environment drop-down menu, the members will only see the environments that they are on the membership list.

Admins can add other users to be an admin of Rancher. They can change a user's role on the **Admin** > **Accounts** page after the user has logged into Rancher. In the **Admin** > **Accounts** tab, click **Edit** next to the account name and change the account type to *Admin*. Click **Save**.

Users

Besides the user that authenticates Rancher, any other user will automatically be added with user permissions. They will not be able to see the **Admin** tab.

They will only be able to view the environments that they are members of.

Disabling Access Control

If you decide that you no longer want Access Control, click the **Disable access control** button. This will make your Rancher instance open to the public and anyone can access your API. This is **not** recommended.

设置

In the **Admin** -> **Settings** page of Rancher, we allow customization of Rancher for different areas of the product.

Host Registration

Before launching any hosts, you will be asked to complete the host registration. This registration sets up how your Rancher server is going to connect with your hosts. If you have already set up access control, you will not be prompted to set host registration as Rancher assumes that your URL will be accessible.

The setup determines the base URL your hosts will use to connect to the Rancher API. By default, Rancher selects the base URL you used to access the UI. If you choose to change the address, make sure to specify the port that should be used to connect to the Rancher API. If you are configuring Rancher with SSL, be sure to change the protocol to `https`. This registration setup determines what the command will be for adding custom hosts.

If access control is turned on for Rancher, only the **admin** will be able to change the host registration. By default, the first **admin** is the first user to authenticate with Rancher when access control was configured. If access-control is still not configured, any users to the site can update the host registration. This option can be updated in the **Admin** -> **Host Registration** tab.

Catalog

By default, the catalog is enabled with two catalogs, the certified templates from the official Rancher Catalog and the community-catalog. You can select to enable or disable these catalogs.

An admin has the ability to add private catalogs to Rancher. Adding a catalog is as simple as adding a catalog name and the git URL. The correct format of the git URL can be found here. Whenever you add a catalog, it will be immediately available in the catalog.

If you want to create your own private catalog to add, the git repository must be set up in a specific format.

Machine Drivers

Docker-machine drivers can be added into Rancher to add hosts.

Adding Machine Drivers

Click on **Add Machine Driver**.

1. Provide a **name** for the driver to be displayed when adding [other hosts]/rancher/rancher-ui/infrastructure/hosts/other/).
2. Provide the **Download URL**. This URL is the machine driver binary 64-bit Linux.
3. (Optional) Provide the **MD5 Checksum** to verify the downloaded driver matches the expected checksum.
4. When complete, click **Create**.

After clicking on create, Rancher add the additional driver and will display this option in the **Driver** field of adding [other hosts]/rancher/rancher-ui/infrastructure/hosts/other/).

账户

环境

API & Keys

点击 **API** 以便找到 API 端点。每当你创建了一个工作环境的 API key，提供的 URL 端点都会将你指向你当前正在操作的那个环境.

如果权限控制未配置，任何知道IP地址的人都拥有访问你的 Rancher API的权限。高度建议启用权限控制管理。

一旦启用了权限控制并且未登录，对每个环境均需要创建一个工作环境 API key，以便 API 能访问指定的工作环境。

在 Rancher 里，通过点击对象的下拉菜单中的 **View In API** ，所有的对象都能通过API来查看。在创建环境 API key 时提供的 URL 端点，同样提供了到 API 各个部分的全部链接。点击此处了解更多关于如何使用 API

添加环境 API Keys

在添加任何环境 API key之前，请确认你在正确的环境中。点击 **Add Environment API Key**。 提供 **Name** ，并根据需要，填写 **Description**。点击 **Create**。Rancher 将会生成并显示你的环境 API Key。一个 API Key 是一个用户名 (access key) 和一个密码 (secret key) 的组合 - 当执行 API 调用时，两个都需要用来进行认证。在你复制保存下相关信息后，点击 **Close**。

> 注意: 一旦你关闭了窗口，你将无法重新获得你的 API Key的密码 (secret key) ，所以确保将其保存在了足够安全的地方。

使用环境 API Keys

当配置了权限控制且你没有登录，如果你试图打开 API 端点，你会被提示需要一个环境 API Key。用户名是access key ，密码是 secret key。

如果你使用 `cURL` ，你能通过指定 `-u` 参数，按照格式 `username:password` 来使用环境 API Key，或者能通过在你的 `.netrc` 文件内加入一行。

编辑环境 API Keys

环境 API Key 的全部选项都可以通过位于 Key 列表右侧的下拉菜单进入。

对于任何 *Active* 的 key，你能 **Deactivate** 这个 key，那将阻止这些证书的使用。Key将会标记为 *Inactive* 状态。

对于任何 *Inactive* 的 key，你有两种选项。 你能 **Activate** 这个 key，这将重新允许证书能访问 API。或者，你能 **Delete** 这个 key, 那将从环境中移除这些证书。

你能 **Edit** 任何 key，那允许你修改这个环境 API key 的名称和描述。你没法修改实际的 access key 或 secret key。如果你想要新的键值对，你需要创建一个新的。

镜像仓库

-- 在 Rancher 中，您可以把 DockerHub、Quay.io 或者任何私有镜像仓库的访问账号添加进来，来访问私有镜像。通过有了访问您的私有镜像库的功能，它是 Rancher 可以去使用您的私有镜像。这样让从一个私有地址中加载一个镜像变得很简单。在 每一个环境中，每一个镜像仓库的地址对一个的访问账号信只能有一条，Rancher 将总是使用最近添加的哪一个。

> **注意:** 目前，镜像仓库只在 **Cattle** 的环境中被支持。

添加镜像仓库

点击 **Infrastructure** 然后进入 **Registries** 页面，点击 **Add Registry**。

对于所有的镜像仓库，您将需要输入 **e-mail address**, **username**, 和 **password** 等信息。对于一个 **Custom** 镜像仓库，您还将需要提供 **registry address**。然后点击 **Create**.

如果您所添加的访问账号信息的地址已经存在了，Rancher 将会开始使用这个新添加的。

非加密镜像仓库

为了访问一个非加密镜像仓库，你讲需要配置所有主机的 Docker 后台访问。 `DOMAIN` 和 `PORT` 是提供私有镜像仓库服务主机的域名和端口。

> **注意:** 任何石油重启主机的 Docker 服务，您都可能会语调网络代理被卡在 *Starting* 状态的问题。为了临时解决这个问题，请重启主机。

```
# 编辑配置文件 "/etc/default/docker"
$ sudo vi /etc/default/docker
# 在文件结束处添加添加下面这行。如果已经有这行了，则确保把下面的参数加载最后。
$ DOCKER_OPTS="$DOCKER_OPTS --insecure-registry=${DOMAIN}:${PORT}"
# 重启 Docker 服务
$ sudo service docker restart
```

自签名证书

为了与私有镜像仓库使用一个自签发证书。 `DOMAIN` 和 `PORT` 是私有镜像仓库服务所在的地址。

```
# 从该域名下载证书
$ openssl s_client -showcerts -connect ${DOMAIN}:${PORT} </dev/null 2>/dev/null|openss
l x509 -outform PEM >ca.crt
# 复制这个证书文件到对应的目录
$ sudo cp ca.crt /etc/docker/certs.d/${DOMAIN}/ca.crt
# 附加这个证书到一个文件内容末尾
$ cat ca.crt | sudo tee -a /etc/ssl/certs/ca-certificates.crt
# 重启 Docker 服务使之生效
$ sudo service docker restart
```

使用镜像仓库

在镜像仓库被创建之后，你将能在启动服务或者容器的时候使用这些私有镜像仓库。镜像名称的语法是与您在 `docker run` 命令中是相同的。

`[registry-name]/[namespace]/[imagename]:[version]`

默认情况下，我们假设您总是从 `DockerHub` 来下拉镜像。

编辑镜像仓库

在镜像仓库清单中的

All options for a registry are accessible through the dropdown menu on the right hand side of the listed registry.

For any **Active** registry, you can **Deactivate** the registry, which would prohibit access to the registry. No new containers can be launched with any images in that registry.

For any **Deactivated** registry, you have two options. You can **Activate** the registry, which will allow containers to access images from those registries. Any members of your environment will be able to activate your credential without needing to re-input the password. If you don't want anyone using your credential, you should **Delete** the registry, which will remove the credentials from the environment.

You can **Edit** any registry, which allows you to change the credentials to the registry address. You will not be able to change the registry address. The password is not saved in the "Edit" page, so you will need to re-input it in order to save any changes.

> **Note:** If a registry is invalid (i.e. inactive, removed, or overridden due to a newer credential), any service or container using a private registry image will continue to run. Since the image has already been pulled onto the host, there will be no restrictions on usage of the image regardless of registry permissions. Therefore, any scaling up of services or additional containers using the image will be able to run. Rancher does not check if the credentials are still valid when running containers as we assume that you've already given the host permissions to access the image.

使用 Docker 原生命令行

Rancher与Docker原生命令行的集成，使它能够轻易地与其他DevOps、Docker工具结合使用。在更高一个层面来说，在Rancher的外部去启动，停止或者销毁容器，Rancher也将会检测到这些容器的变化，并将其更新到对应的状态

Docker 事件监控

Rancher通过监听Docker events来更新所有主机的容器状态。当在Rancher平台之外启动、停止或者销毁容器（比如说，直接在主机上执行 `docker stop sad_einstein` ），Rancher将会检测到这些容器的所发生的变化并将其更新到相应的状态。

> 注意： 当前的一个限制是：Rancher会等待容器已经启动了（不是创建）才会将其导入到Rancher，执行 `docker create ubuntu` 会导致这个容器在Rancher UI上消失，但是执行 `docker start ubuntu` 或者 `docker run ubuntu` 则不会。

Rancher通过 `docker events` 命令去监控一个主机上容器的变化，你也可以试试使用这个命令去观察Docker的事件流。

将已启动的容器加入到Rancher中

你可以不通过Rancher去启动容器，并能让Rancher去控制这些容器。这意味着容器可以运行在跨主机的网络中。在创建容器的时候添加一个值为 `true` 的 `io.rancher.container.network` 标签来启用这个特性。

```
$ docker run -l io.rancher.container.network=true -itd ubuntu bash
```

如需了解更多关于Rancher管理网络和跨主机网络，请阅读Rancher基本概念.

导入已存在的容器

Rancher还支持直接导入主机上已经存在的容器。你可以参考添加自己的主机添加一个自己的主机，添加之后会发现主机上所有的容器都会被检测到并导入Rancher中。

定期同步状态

除了实时监听Docker事件外，Rancher还定期的去同步主机的状态。Rancher每隔5秒钟会去收集主机所有的容器信息，以确保平台上所展示的容器状态和实际的主机上的状态相符。防止由于网络中断或者服务器重启导致Rancher没接受到Docker事件。使用这种方式去同步状

态，Rancher上的容器状态将会和主机上的容器状态保持一致。比如说，Rancher觉得一个容器正在运行中，但它在主机上实际上是被停止了，Rancher将会把该容器的状态更新成停止状态，而不是去尝试重启这个容器。

使用 Rancher-Compose

Jitao Hou 正在翻译首稿，欢迎之后Review

`Rancher-compose` 工具就像一个多主机版本的 `docker-compose` . 它直接操作Rancher图形界面中的 应用栈（stack）, 应用栈属于一个环境（environment），并且包含很多主机（hosts）. 由 `rancher-compose` 启动的容器将会被部署在所属环境内的任意满足调度规则（scheduling rules）的主机节点上. 如果事先没有定义调度规则， 服务中的容器将会在已启动容器数目较少的节点上来启动。 因为使用相同的API调用， 使用ancher-compose就和用户界面中启动服务的效果一样，服务中的容器将会被拉起。

`Rancher-compose` 工具就像大家熟知的 `docker-compose` 一样， 支持任意的 `docker-compose.yml` 文件. 另外，它还支持 `rancher-compose.yml` 文件，该文件扩充和覆写了 `docker-compose.yml` 的定义，包含了很多只有在Rancher环境才被支持的属性，如， 服务中容器的启动数量等。

如需深入掌握本章关于 `rancher-compose` 的知识，用户需要对 `docker-compose` 有基本的了解。请在开始使用 `rancher-compose` 之前，通读docker-compose documentation 。

安装

该命令的二进制文件可以直接在用户界面（UI）中下载，下载的链接位于UI的右下角，其中包含支持Windows, Mac和 Linux的命令行文件.

你也可以参考releases page for rancher-compose 来直接下载该工具。

在Rancher服务器中设置Rancher-Compose

为了使得 `rancher-compose` 能够在Rancher的实例中启动服务，你需要设置一些环境变量，或者在 `rancher-compose` 命令行中以选项的方式传递这些变量。这些所需要的环境变量是 `RANCHER_URL` , `RANCHER_ACCESS_KEY` , 和 `RANCHER_SECRET_KEY` . Access key和Secret key属于API key.

```
bash
# Set the url that Rancher is on
$ export RANCHER_URL=http://server_ip:8080/
# Set the access key, i.e. username
$ export RANCHER_ACCESS_KEY=<username_of_key>
# Set the secret key, i.e. password
$ export RANCHER_SECRET_KEY=<password_of_key>
```

如果你选择不设置这些环境变量，你将会需要在 `rancher-compose` 命令行中以选项的方式传递这些变量。

```
bash
$ rancher-compose --url http://server_ip:8080 --access-key <username_of_key> --secret-key <password_of_key> up
```

现在, 你可以利用 `rancher-compose` 搭配任意的 `docker-compose.yml` 文件来启动服务. 该服务将会在API Key所在的environment中的相关Rancher 实例里自动被启动。

命令

如需了解关于命令和选项的更多信息，请参考 rancher-compose command 文档.

命令举例

```
bash
# Creating and starting a service without environment variables and picking a stack
$ rancher-compose --url URL_of_Rancher --access-key username_of_API_key --secret-key password_of_API_key -p stack1 up
# To change the scale of an existing service
$ rancher-compose -p stack1 scale web=3
```

删除服务／容器

在缺省情况下, `rancher-compose` 不会删除服务／容器。这意味着如果你在一行内有两个 `up` 命令，第二个 `up` 将不起作用。这是因为第一个up指令将会创建每个对象，并且让它运行。 即使你不传递 `-d` 选项给 `up` 指令, `rancher-compose` 也不会删除你的服务。你必须使用 `rm` 来删除一个服务。

Build

有两种方式来支持 docker build。第一种是设置 `build:` 到一个与Docker Remote API 中的远程参数相兼容的 git 地址或 HTTP URL。第二种方法是设置 `build:` 到一个本地目录，build的上下文将会被上传到S3，然后在每个节点上按指令构建。

为了基于S3 的build正常工作，你必须设置AWS credentials。 我们提供了如何在rancher-compose使用S3的详细样例

Sidekicks

使用服务时， 你可能需要你所定义的服务使用 `volumes_from` 和 `net` 来指向另外一个服务. 为了达到这个目的，你需要设置sidekick关系。 Sidekick关系使得Rancher能够像一个单元一样来伸缩和调度一组服务。举例: B是A的sidekick, 这两个服务将会被始终成对部署，服务中的容器数量也将始终是一样的。

另外你需要定义sidekick的情形是，你需要确保多个服务始终部署在同一个主机节点上。

为了设置sidekick关系, 你需要为其中一个服务加上标签（label）。标签的关键字是 `io.rancher.sidekicks` ，其赋值是一个服务或多个服务名称。如果你需要添加多个服务作为sidekick，这些服务名称需要用逗号隔开。举例: `io.rancher.sidekicks: sidekick1, sidekick2, sidekick3`

为一个服务定义了sidekick之后，你将不需要在 `rancher-compose` 里设置link，因为这些这些在同一环境的服务将会自动被DNS解析到。

主服务

包含sidekick的服务被称之为主服务,其sidekick将被认为是从服务。主服务的尺寸（容器多少）将会被应用于所有在sidekick的从服务。 如果你所有服务的尺寸不一致，主服务的尺寸将会被用在所有服务上。

当为定义了sidekick的服务配置负载均衡器 时，你只能为主服务设置目标服务。Sidekick服务不能够 被定义为目标服务。

Rancher-Compose中的Sidekick举例:

Sample configuration `docker-compose.yml`

```
yaml
test:
  tty: true
  image: ubuntu:14.04.2
  stdin_open: true
  volumes_from:
  - test-data
  labels:
    io.rancher.sidekicks: test-data
test-data:
  tty: true
  command:
  - cat
  image: ubuntu:14.04.2
  stdin_open: true
```

Sample `rancher-compose.yml`

```
test:
  scale: 2
test-data:
  scale: 2
```

Rancher-Compose中的Sidekick举例：多个服务使用同一个 `volumes_from`

如果你有多个服务使用同一个容器来做 `volumes_from` , 你可以为主服务增加一个定义为sidekick的从服务，来使用同一个数据容器。因为只有一个容器能够被作为负载均衡的目标，所以请确保选择一个正确的服务作为主服务(即定义了sidekick标签的服务)。

```
yaml
test-data:
  tty: true
  command:
  - cat
  image: ubuntu:14.04.2
  stdin_open: true
test1:
  tty: true
  image: ubuntu:14.04.2
  stdin_open: true
  labels:
    io.rancher.sidekicks: test-data, test2
  volumes_from:
  - test-data
test2:
  tty: true
  image: ubuntu:14.04.2
  stdin_open: true
  volumes_from:
  - test-data
```

为服务添加链接（link）

在Rancher中，同一个环境里的所有服务都是可以被DNS解析到的，所以并不需要为服务显示地添加链接（link），除非你需要使用一个指定的别名来做DNS解析。

> **Note:** 目前我们不支持将sidekick链接到主服务或者主服务连接到sidekick。更多信息，请参考Rancher内置DNS的工作原理.

属于同一个应用栈的服务，任何服务可以按照自己原有的 `服务名称` 被DNS解析到, 当然，你可以定义链接，如果你希望为某一个服务定义别名的话。

`docker-compose.yml` 的一个例子

```
wordpress:
  image: wordpress
  links:
  - db:mysql
db:
  image: mysql
```

在这个例子中, `db` 将被解析为 `mysql`. 如果没有定义链接（link）, `db` 将只能按 `db` 解析.

在其它应用栈中的服务，服务将会按照 `service_name.stack_name` 的格式被DNS解析。 如果你需要使用一个指定的别名做DNS解析，你需要在 `docker-compose.yml` 中定义 `external_links` 。

`docker-compose.yml` 的一个例子

```
wordpress:
  image: wordpress
  external_links:
    - alldbs/db1:mysql
```

在这个例子中, `alldbs` 应用栈中有一个 `db1` 服务，wordpress将会链接到它。 在 `wordpress` 服务中, `db1` 将会被解析为 `mysql`. 如果没有定义external link, `db1` 将按 `db1.alldbs` 解析.

> 备注: 跨应用栈的服务发现只限于同一个环境之内(by design)，不支持跨环境的服务发现。

使用 Rancher Compose

命令和选项

Jitao Hou 正在翻译首稿，欢迎之后Review

The `rancher-compose` tool works just like the popular `docker-compose` and supports any `docker-compose.yml` file. There is also a `rancher-compose.yml` which extends and overwrites `docker-compose.yml.` The rancher-compose yaml file are attributes only supported in Rancher, for example, scale of a service.

Rancher-Compose Commands

`rancher-compose` supports any command that `docker-compose` supports.

Name	Description
`create`	Create all services but do not start
`up`	Bring all services up
`start`	Start services
`logs`	Get service logs
`restart`	Restart services
`stop` , `down`	Stop services
`scale`	Scale services
`rm`	Delete services
`pull`	Pulls images for services
`upgrade`	Perform rolling upgrade between services
`help` , `h`	Shows a list of commands or help for one command

Rancher-Compose Options

Whenever you use the `rancher-compose` command, there are different options that you can use.

Name	Description
`--verbose` , `--debug`	
`--file` , `-f` "docker-compose.yml"	Specify an alternate compose file (default: docker-compose.yml) [$COMPOSE_FILE]
`--project-name` , `-p`	Specify an alternate project name (default: directory name)
`--url`	Specify the Rancher API endpoint URL [$RANCHER_URL]
`--access-key`	Specify Rancher API access key [$RANCHER_ACCESS_KEY]
`--secret-key`	Specify Rancher API secret key [$RANCHER_SECRET_KEY]
`--rancher-file` , `-r`	Specify an alternate Rancher compose file (default: rancher-compose.yml)
`--env-file` , `-e`	Specify a file from which to read environment variables
`--help` , `-h`	show help
`--version` , `-v`	print the version

Examples

```
# Starting a service without environment variables and defining a stack name
$ rancher-compose --url URL_of_Rancher --access-key username_of_API_key --secret-key p
assword_of_API_key -p stack1 up
# To change the scale of an existing service
$ rancher-compose -p stack1 scale web=3
```

> **Note:** If you don't pass in `-p STACK_NAME`, the stack name will be the directory that you are running the `rancher-compose` command in.

Command Options

Up Command

Name	Description
`--pull` , `-p`	Before doing the upgrade do an image pull on all hosts that have the image already
`-d`	Do not block and log
`--upgrade` , `-u` , `--recreate`	Upgrade if service has changed
`--force-upgrade` , `--force-recreate`	Upgrade regardless if service has changed
`--confirm-upgrade` , `-c`	Confirm that the upgrade was success and delete old containers
`--rollback` , `-r`	Rollback to the previous deployed version
`--batch-size "2"`	Number of containers to upgrade at once
`--interval "1000"`	Update interval in milliseconds

When you run the `up` command with `rancher-compose` , after all the tasks are complete, the process continues to run. If you want the process to exit after completion, you'll need to add in the `-d` option, which is to not block and log.

```
# If you do not use the -d flag, rancher-compose will continue to run until you Ctrl+C
 to quit.
$ rancher-compose up
# Use the -d for rancher-compose to exit after running
$ rancher-compose up -d
```

Read more about upgrading using rancher-compose.

Start Command

Name	Description
`-d`	Do not block and log

If you want the start process to exit after completion, you'll need to add in the `-d` option, which is to not block and log.

Logs Command

Name	Description
`--lines "100"`	number of lines to tail

Restart Command

Name	Description
`--batch-size` `"1"`	Number of containers to retart at once
`--interval` `"0"`	Restart interval in milliseconds

By default, restarting services will restart the containers individually and immediately sequentially. You can set the batch size and interval of the restart policy.

Stop/Down & Scale Command

Name	Description
`--timeout` , `-t` `"10"`	Specify a shutdown timeout in seconds.

Rm Command

Name	Description
`--force` , `-f`	Allow deletion of all services

Pull Command

Name	Description
`--cached` , `-c`	Only update hosts that have the image cached, don't pull new

```
# Pulls new images for all services located in the docker-compose.yml file on ALL host
s in the environment
$ rancher-compose pull
# Pulls new images for all services located in docker-compose.yml file on hosts that a
lready have the image
$ rancher-compose pull --cached
```

> **Note:** Unlike `docker-compose pull`, you will not be specifying which service to pull. Rancher-compose looks at all services in the `docker-compose.yml` and pulls images for all services found in the file.

Upgrade Command

Rancher supports upgrades to services using `rancher-compose`. Please read more about when and how to upgrade your services.

Name	Description
`--batch-size "2"`	Number of containers to upgrade at once
`--scale "-1"`	Final number of running containers
`--interval "2000"`	Update interval in milliseconds
`--update-links`	Update inbound links on target service
`--wait`, `-w`	Wait for upgrade to complete
`--pull`, `-p`	Before doing the upgrade do an image pull on all hosts that have the image already
`--cleanup`, `-c`	Remove the original service definition once upgraded, implies `--wait`

```
# Upgrade service1 to service2
# service1 and service2 to be defined in the docker-compose.yml
$ rancher-compose upgrade service1 service2

# Upgrade to a different scale
$ rancher-compose upgrade service1 service2 --scale 5

# Removes service1 from Rancher
$ rancher-compose upgrade service1 service2 --cleanup
```

Compose Compatibility

`rancher-compose` strives to be completely compatible with Docker Compose. Since `rancher-compose` is largely focused on running production workloads some behaviors between Docker Compose and Rancher Compose are different.

We support anything that can be created in a standard docker-compose.yml file. There are a couple of differences in the behavior of rancher-compose that are documented below.

Deleting Services/Container

`rancher-compose` will not delete things by default. This means that if you do two `up` commands in a row, the second `up` will do nothing. This is because the first up will create everything and leave it running. Even if you do not pass `-d` to `up`, `rancher-compose` will not delete your services. To delete a service you must use `rm`.

Rancher 服务

Jitao Hou 正在翻译首稿，欢迎之后Review

Rancher implements a distributed health monitoring system by running an HAProxy instance on every host for the sole purpose of providing health checks to containers. When health checks are enabled either on an individual container or a service, each container is then monitored by up to three HAProxy instances running on different hosts. The container is considered healthy if at least one HAProxy instance reports a "passed" health check and it is considered unhealthy when all HAProxy instances report a "unhealthy" health check.

Rancher's approach handles network partitions and is more efficient than client-based health checks. By using HAProxy to perform health checks, Rancher enables users to specify the same health check policy for DNS service and for load balancers.

> **Note:** Health checks will only work for services that are using the managed network. If you select any other network choice, it will **not** be monitored.

To enable health checks for services, we add the health check in the `rancher-compose.yml` file.

```
wordpress:
  scale: 1
  health_check:
    port: 80
    # Interval is measured in milliseconds
    interval: 2000
    unhealthy_threshold: 3
    # For TCP, request_line needs to be ''
    # TCP Example:
    # request_line: ''
    request_line: GET /healthcheck HTTP/1.0
    healthy_threshold: 2
    # Response timeout is measured in milliseconds
    response_timeout: 2000
```

More details about Health Checks can be read in the concept section.

> **Note:** If a host is down in Rancher (i.e. in `reconnecting` or `inactive` state), you will need to implement a health check in order for Rancher to launch the containers on your service on to a different host.

Custom Rancher Services

Custom Rancher services are configured by using a special image name in the compose template. The image name is how `rancher-compose` knows to set up a Rancher service versus a normal service.

Service	Image Name
Load Balancer	rancher/load-balancer-service
External Service	rancher/external-service
Alias/DNS Service	rancher/dns-service

Load Balancer

A load balancer can be scheduled like any other service. Read more about scheduling services and load balancers using `rancher-compose`.

Rancher supports L4 load balancing by adding ports and linking target services. Any traffic directed to any of source port(s) will be sent to the private port(s) of the linked service(s).

> **Note:** Port `42` cannot be used as a source port for load balancers because it's internally used for health checks.

When working with services that contains sidekicks, you need to link the primary service, which is the service that contains the `sidekick` label.

> **Note:** Load balancers will only work for services that are using the managed network. If you select any other network choice for your target services, it will **not** work with the load balancer.

Load Balancer Example (L4)

Sample configuration `docker-compose.yml`

```
lb:
  image: rancher/load-balancer-service
  ports:
  # Assign a random public port and direct traffic to private port 80 of the service.
  - 80
  # Listen on public port 80 and direct traffic to private port 80 of the service
  - 80:80
  # Listen on public port 82 and by default forward traffic to private port 81 using H
TTP
  - 82:81
  # Assign a random public port and direct traffic to private port 9999 using TCP and
not HTTP
  - 9999/tcp
  links:
  # Target services in the same stack will be listed as a link
  - web1:web1
  - web2:web2
  external_links:
  # Target services in a different stack will be listed as an external link
  - differentstack/web3:web3
```

Sample `rancher-compose.yml`

```
lb:
  # Two load balancer containers
  scale: 2
  load_balancer_config:
    name: lb config
web1:
  scale: 1
web2:
  scale: 1
web3:
  scale: 1
```

Advanced Load Balancing (L7)

We also support L7 load balancing with advanced routing options, which include using host headers, host paths and specific target ports. We go into more detail on advanced routing options for load balancers in our UI section, but anything that we can create in Rancher can be created using `rancher-compose`.

Advanced routing options use `labels` in the `docker-compose.yml` file. Here's the basic syntax for applying the label. You would only use this label if you were going to do some advanced routing rules. Advanced routing rules are optional and all of the fields are optional as well. This syntax shows if you were to use all the options.

Label Key	Label Value
`io.rancher.loadbalancer.target.<SERVICE_NAME>`	`<REQUEST_HOST>:<SOURCE_PORT>/<REQUEST_PATH>=<TARGET_PORT>`

> **Note:** If you use a source port with the advanced routing options, the source port must also be listed in `ports` section of the `docker-compose.yml`

Linking Services in a Different Stack

In the label for target services, `<SERVICE_NAME>` is the name of the service. If your service is in another stack, the `<SERVICE_NAME>` will need to include stack name using the following format `<STACK_NAME>/<SERVICE_NAME>` . The link to the target service of other stacks will be under `external_links` .

Syntax of the Combination of all Optional Fields

Since the fields are optional, we support all combinations of the fields. Here is the syntax for all combinations using request host, source port, path and target port.

Request Host	Source Port	Path	Target Port	Label Value
example.com	80	/path	81	`example.com:80/path=81`
example.com	80	/path/a		`example.com:80/path/a`
example.com	80		81	`example.com:80=81`
example.com	80			`example.com:80`
example.com		/path/b/c	81	`example.com/path/b/c=81`
example.com		/path		`example.com/path`
example.com			81	`example.com=81`
example.com				`example.com`
	80	/path	81	`80/path=81`
	80	/path		`80/path`
	80		81	`80=81`
		/path	81	`/path=81`
		/path		`/path`
			81	`'81'` *See Note

> **Note:** It is assumed that if you have only a port in the label, then the port is for the target port of the service. When using only a target port, it must be surrounded by single quotes.

Wildcards

Rancher supports wildcards when adding host based routing. The following wildcard syntax is supported.

```
*.domain.com -> hdr_end(host) -i .domain.com
domain.com.* -> hdr_beg(host) -i domain.com.
```

Multiple Routing Rules for the Same Service

In `rancher-compose`, you can configure multiple hostname routing rules to the same service by separating each rule with a comma. See the example for service `web2`.

Priority

When there are multiple hostname routing rules, the order of priority is as follows:

1. Hostname and URL
2. Hostname only
3. URL
4. Default (no hostname, no URL)

Load Balancer Example (L7)

Sample configuration `docker-compose.yml`

```
lb:
  image: rancher/load-balancer-service
  ports:
  # Listen on public port 80 and direct traffic to private port 80 of the service
  - 80:80
  # Listen on public port 82 and by default forward traffic to private port 81 using H
TTP
  - 82:81
  # Listen on public port 9999 using TCP and not HTTP
  - 9999/tcp
  links:
  # Target services in the same stack will be listed as a link
  - web1:web1a
  - web2:web2a
  external_links:
  # Target services in a different stack will be listed as an external link
  - differentstack/web3:web3a
  labels:
    # Put load balancer containers on hosts with label lb=true
    io.rancher.scheduler.affinity:host_label: lb=true
    # Requests to http://app.example.com/foo:80 should be routed to web1 over port 8000

    io.rancher.loadbalancer.target.web1: app.example.com:80/foo=8000
    # Requests to http://app.example.com/foo should be routed to web2 over port 8000
    # and http://app.example.com/foo/bar over port 8001
    io.rancher.loadbalancer.target.web2: app.example.com/foo=8000,app.example.com/foo/
bar=8001
    # Requests routed to web3 go to port 8000, overriding the default configuration
    # of 82:81
    io.rancher.loadbalancer.target.differentstack/web3: 82=8000
```

Sample `rancher-compose.yml`

```
lb:
  # Two load balancer containers
  scale: 2
  load_balancer_config:
    name: lb config
web1:
  scale: 1
web2:
  scale: 1
web3:
  scale: 1
```

Internal Load Balancer

To set an internal load balancer, the listening ports are listed under `expose` instead of `ports`.

Sample configuration `docker-compose.yml`

```
lb:
  image: rancher/load-balancer-service
  # Instead of using ports, we use expose to define that it will be private ports
  expose:
  # Listen on private port 80 and direct traffic to private port 80 of the service
  - 80:80
  # Listen on private port 82 and by default forward traffic to private port 81 using
HTTP
  - 82:81
  links:
  # Any service that is a target will be listed as a link
  - web1:web1
```

Sample `rancher-compose.yml`

```
lb:
  # Two load balancer containers
  scale: 2
  load_balancer_config:
    name: lb config
web1:
  scale: 1
```

Custom haproxy.cfg

For advanced users, you can specify `global` and `defaults` configuration to the load balancer in the `rancher-compose.yml`. Please refer to the HAProxy documentation for details on the available options you can add.

Sample `rancher-compose.yml`

```
lb:
  scale: 1
  load_balancer_config:
    haproxy_config:
      defaults: <DEFAULTS_INPUTS>
      global: <GLOBAL_INPUTS>
  health_check:
    port: 42
    interval: 2000
    unhealthy_threshold: 3
    healthy_threshold: 2
    response_timeout: 2000
```

External Service

With external services, you can set either external IP(s) **OR** a domain name. The `rancher/external-service` is not an actual image, but is required for the docker-compose.yml. There are no containers created for external services.

Sample configuration `docker-compose.yml`

```
db:
  image: rancher/external-service

redis:
  image: redis
```

Sample `rancher-compose.yml` using external IPs

```
db:
  external_ips:
  - 1.1.1.1
  - 2.2.2.2

# Override any service to become an external service
redis:
  image: redis
  external_ips:
  - 1.1.1.1
  - 2.2.2.2
```

Sample `rancher-compose.yml` using hostname

```
db:
  hostname: example.com
```

Service Alias/Internal DNS service

A service alias creates a pointer to service(s). In the example below, `web[.stack-name.rancher.internal]` will resolve to the IPs of the containers of `web1` and `web2`. The `rancher/dns-service` is not an actual image, but is required for the docker-compose.yml. There are no containers created for alias services.

Sample configuration `docker-compose.yml`

```
web:
  image: rancher/dns-service
  links:
  - web1
  - web2

web1:
  image: nginx

web2:
  image: nginx
```

调度

Jitao Hou 正在翻译首稿，欢迎之后Review

When creating services using `rancher-compose`, you can direct the host(s) of where the containers should be launched based on scheduling rules. This scheduling ability is available in the UI and also support in `rancher-compose`. Rancher determines how to schedule a service's containers based on the `labels` defined in the `docker-compose.yml` file.

Labels used in Docker-Compose

All of the labels in this section would be used in the `docker-compose.yml` file. Rancher defines the scheduling rules with 3 main components: conditions, fields and values. Conditions determine how strictly Rancher follows the rules. Fields are which items that are going to be compared against. Values are what you've defined on the fields. We'll talk broadly about these components before going into some examples.

Scheduling Conditions

When we write our scheduling rules, we have conditions for each rule, which dictates how Rancher uses the rule. An affinity condition is when we are trying to find a field that matches our value. An anti-affinity condition is when we are trying to find a field that does not match our value.

To differentiate between affinity and anti-affinity, we add `_ne` to the label name to indicate that the label is **not** matching the field and values.

There are also hard and soft conditions of a rule.

A hard condition is the equivalent of saying **must** or **must not**. Rancher will only use hosts that match or do not match the field and value. If Rancher cannot find a host that meets all of the rules with these conditions, your service could get stuck in an *Activating* state. The service will be continually trying to find a host for the containers. To fix this state, you can either edit the scale value of the service or add/edit another host that would satisfy all of these rules.

A soft condition is the equivalent of saying **should** or **should not**. Rancher will attempt to use hosts that match the field and value. In the case of when there is no host that matches all the rules, Rancher will remove one by one the soft constraints (should/should not rules) until a host satisfies the remaining constraints.

To differentiate between the *must* and *should* conditions, we add `_soft` to our label name to indicate that the label is **should** try to match the field and values.

Fields

Rancher has the ability to compare values against host labels, container labels, container name, or service name. The label prefix is what Rancher uses to define which field will be evaluated.

Field	Label Prefix
Host Label	`io.rancher.scheduler.affinity:host_label`
Container Label/Service Name	`io.rancher.scheduler.affinity:container_label`
Container Name	`io.rancher.scheduler.affinity:container`

Notice how there is not a specific prefix for service name. When Rancher creates a service, system labels are added to all containers of the service to indicate the stack and service name.

To create the key of our label, we start with a field prefix (e.g. `io.rancher.scheduler.affinity:host_label`) and based on the condition that we are looking for, we append the type of condition we want. For example, if we want the containers to be launched on a host that must not equal (i.e. `_ne`) to a host label value, the label key would be `io.rancher.scheduler.affinity:host_label_ne` .

Values

You use the values to define what you want the field to be checked against. If you have a couple of values that you want to compare against for the same condition and field, you'll need to use only one label for the name of the label. For the value of the label, you'll need to use a comma separated list. If there are multiple labels with the same key (e.g. `io.rancher.scheduler.affinity:host_label_ne`), Rancher will overwrite any previous value with the last value that is used with the label key.

```
labels:
  io.rancher.scheduler.affinity:host_label: key1=value1,key2=value2
```

Global Service

Making a service into a global service is the equivalent of selecting **Always run one instance of this container on every host** in the UI. This means that a container will be started on any host in the [environment]/rancher/configuration/environments/). If a new host

is added to the environment, and the host fulfills the global service's host requirements, the service will automatically be started.

Currently, we only support global services with host labels fields that are using the hard condition. This means that only labels that are related to `host_labels` will be adhered to when scheduling and it **must** or **must not** equal the values. Any other label types will be ignored.

Example `docker-compose.yml` :

```
wordpress:
  labels:
    # Make wordpress a global service
    io.rancher.scheduler.global: 'true'
    # Make wordpress only run containers on hosts with a key1=value1 label
    io.rancher.scheduler.affinity:host_label: key1=value1
    # Make wordpress only run on hosts that do not have a key2=value2 label
    io.rancher.scheduler.affinity:host_label_ne: key2=value2
  image: wordpress
  links:
    - db: mysql
  stdin_open: true
```

Finding Hosts with Host Labels

When adding hosts to Rancher, you can add host labels. When scheduling a service, you can leverage these labels to create rules to pick the hosts you want your service to be deployed on.

Examples for each condition type:

```
labels:
  # Host MUST have the label `key1=value1`
  io.rancher.scheduler.affinity:host_label: key1=value1
  # Host MUST NOT have the label `key2=value2`
  io.rancher.scheduler.affinity:host_label_ne: key2=value2
  # Host SHOULD have the label `key3=value3`
  io.rancher.scheduler.affinity:host_label_soft: key3=value3
  # Host SHOULD NOT have the label `key4=value4`
  io.rancher.scheduler.affinity:host_label_soft_ne: key4=value4
```

Automatically Applied Host Labels

Rancher automatically creates host labels related to linux kernel version and Docker Engine version of the host.

Key	Value	Description
`io.rancher.host.linux_kernel_version`	Linux Kernel Version on Host (e.g, `3.19`)	Version of the Linux kernel running on the host
`io.rancher.host.docker_version`	Docker Engine Version on the host (e.g. `1.10.3`)	Docker Engine Version on the host

```
labels:
# Host MUST be running Docker version 1.10.3
io.rancher.scheduler.affinity:host_label: io.rancher.host.docker_version=1.10.3
# Host MUST not be running Docker version 1.6
io.rancher.scheduler.affinity:host_label_ne: io.rancher.host.docker_version=1.6
```

> **Note:** Rancher does not support the concept of scheduling containers on a host that has `>=` a specific version. You can create specific whitelists and blacklists by using the host scheduling rules to determine if a specific version of Docker Engine is required for your services.

Finding Hosts with Container Labels

When adding containers or services to Rancher, you can add container labels. These labels can be used for the field that you want a rule to compare against. Reminder: This cannot be used if you set global service to true.

> **Note:** If there are multiple values for container labels, Rancher will look at all labels on all containers on the host to check the container labels. The multiple values do not need to be on the same container on a host.

Examples for each condition type:

```
labels:
  # Host MUST have a container with the label `key1=value1`
  io.rancher.scheduler.affinity:container_label: key1=value1
  # Host MUST NOT have a container with the label `key2=value2`
  io.rancher.scheduler.affinity:container_label_ne: key2=value2
  # Host SHOULD have a container with the label `key3=value3`
  io.rancher.scheduler.affinity:container_label_soft: key3=value3
  # Host SHOULD NOT have a container with the label `key4=value4
  io.rancher.scheduler.affinity:container_label_soft_ne: key4=value4
```

Service Name

When `rancher-compose` starts containers for a service, it also automatically creates several container labels. Since checking for a specific container label is looking for a `key=value` , we can use these system labels as the key of our rules. Here are the system labels created on the containers when Rancher starts a service:

Label	Value
io.rancher.stack.name	`$${stack_name}`
io.rancher.stack_service.name	`$${stack_name}/$${service_name}`

> **Note:** When using the `io.rancher.stack_service.name` , the value must be in the format of `stack name/service name` .

The macros `$${stack_name}` and `$${service_name}` can also be used in the `docker-compose.yml` file in any other `label` and will be evaluated when the service is started.

Please note that in versions prior to Rancher v0.41.0 and Rancher-compose v0.4.1, the values had a single `$` , but with the addition of environment interpolation, the values require a double `$$` .

Example of Service Name:

```
labels:
  # Host MUST have a container from service name `value1`
  io.rancher.scheduler.affinity:container_label: io.rancher.stack_service.name=stackna
me/servicename
```

Finding Hosts with Container Names

When adding containers to Rancher, you give each container a name. You can use this name as a field that you want a rule to compare against. Reminder: This cannot be used if you set global service to true.

```
labels:
  # Host MUST have a container with the name `value1`
  io.rancher.scheduler.affinity:container: value1
  # Host MUST NOT have a container with the name `value2`
  io.rancher.scheduler.affinity:container_ne: value2
  # Host SHOULD have a container with the name `value3`
  io.rancher.scheduler.affinity:container_soft: value3
  # Host SHOULD NOT have a container with the name `value4
  io.rancher.scheduler.affinity:container_soft_ne: value4
```

Examples

Example 1:

A typical scheduling policy may be to try to spread the containers of a service across the different available hosts. One way to achieve this is to use an anti-affinity rule to itself:

```
labels:
  io.rancher.scheduler.affinity:container_label_ne: io.rancher.stack_service.name=$${s
tack_name}/$${service_name}
```

Since this is a hard anti-affinity rule, we may run into problems if the scale is larger than the number of hosts available. In this case, we might want to use a soft anti-affinity rule so that the scheduler is still allowed to deploy a container to a host that already has that container running. Basically, this is a soft rule so it can be ignored if no better alternative exists.

```
labels:
  io.rancher.scheduler.affinity:container_label_soft_ne: io.rancher.stack_service.name
=$${stack_name}/$${service_name}
```

Example 2:

Another example may be to deploy all the containers on the same host regardless of which host that may be. In this case, a soft affinity to itself can be used.

```
labels:
  io.rancher.scheduler.affinity:container_label_soft: io.rancher.stack_service.name=$$
{stack_name}/$${service_name}
```

If a hard affinity rule to itself was chosen instead, the deployment of the first container would fail since there would be no host that currently has that service running.

Table of Scheduling Labels

Label	Value
io.rancher.scheduler.global	true
io.rancher.scheduler.affinity:host_label	key1=value1,key2=value2, etc...
io.rancher.scheduler.affinity:host_label_soft	key1=value1,key2=value2
io.rancher.scheduler.affinity:host_label_ne	key1=value1,key2=value2
io.rancher.scheduler.affinity:host_label_soft_ne	key1=value1,key2=value2
io.rancher.scheduler.affinity:container_label	key1=value1,key2=value2

io.rancher.scheduler.affinity:container_label_soft	key1=value1,key2=value2
io.rancher.scheduler.affinity:container_label_ne	key1=value1,key2=value2
io.rancher.scheduler.affinity:container_label_soft_ne	key1=value1,key2=value2
io.rancher.scheduler.affinity:container	container_name1,container_name2
io.rancher.scheduler.affinity:container_soft	container_name1,container_name2
io.rancher.scheduler.affinity:container_ne	container_name1,container_name2

io.rancher.scheduler.affinity:container_soft_ne	container_name1,container_name2

升级服务

环境插值

构建使用 AWS S3存储

Rancher 图像界面

rancher中的标签和调度

当通过服务(service)，或者通过负载均衡器(loadbalancer)，或者通过容器页面(container page)启动容器的时候，我们提供为容器创建标签的选项以及提供调度容器到你想要放置容器的主机上的能力。在这章节剩下的部分，我们将使用术语容器/服务(container/service),但这些标签也适用于负载均衡器(也就是一种特殊类型的服务)。

调度规则为你想要Rancher如何挑选使用的主机提供了灵活性。在Rancher中，我们使用标签来帮助定义调度规则。你能为一个容器创建你想要的数量的标签。有了多种规则，你完全控制哪种主机上创建容器。你可以请求具有特定主机标签，容器标签或名字，或者特定服务的容器在一台主机启动。这些调度规则能帮助创建你容器托管关系的黑名单和白名单。

标签

对于服务/容器, 标签可以在**Advanced Options -> Labels**中找到。对于负载均衡器，标签可以在**Labels**选项卡中找到。

通过添加标签给负载均衡器，每个负载均衡器容器都会收到该标签，这标签是一个键值对。Rancher中我们使用这些容器标签来帮助定义调度规则。你能在负载均衡器上创建你想要的数量的标签。默认地Rancher已经为每个容器添加了系统相关标签。

调度选项

对于服务/容器,标签能在**Advanced Options -> Scheduling**中找到。对于负载均衡器，标签能在**Scheduling**选项卡中找到。

对于服务/容器, 提供了两个选项来决定在哪里启动你的容器。

选项 **1:** 在指定主机上运行所有容器 通过选择该选项运行所有容器在指定的主机，容器/服务将会在指定的主机上启动。如果你的主机停止运行，那么主机上的容器也将停止运行。如果你在容器页面创建一个容器，即使存在端口冲突，容器也能被启动。如果你创建一个规模(scale)大于1的服务，存在端口冲突，你的服务可能陷入未激活状态(inActivating)直到你编辑服务的规模值。

选项 **2:** 自动选择匹配调度规则的主机 通过选择该选项自动选择匹配调度规则的主机，你可以灵活地选择你的调度规则。任何遵守所有规则的主机都是容器能在上面启动的主机。你能通过点击**+**按钮来添加规则。

对于负载均衡器()/rancher/rancher-ui/applications/stacks/adding-balancers/), 因为端口冲突问题只有选项2是有效的。你只能选择添加调度规则。点击**Scheduling**选项卡。你能通过**Add Scheudling Rule**按钮添加你想要的调度规则。

对于每个规则，你要选择规则的条件。有四个不同的条件，分别定义规则必须遵守的规则有多严格。字段**(field)**决定由你希望规则应用于哪些字段。键和值是那些你想要检查的字段的值。如果你启动一个服务或者负载均衡器，rancher将根据每台主机上的负载来传播容器的分布到合适的主机上。决定哪些主机合适取决于条件的选择。

> 注意：对于服务/负载均衡器，如果你选择了规模为总是在每台主机上运行这个容器的一个实例选项，那么只有主机标签作为一个可能的字段出现。

条件

- 必须或必须不：Rancher只使用匹配或者不匹配字段和值的主机。如果Rancher找不到一台主机符合带有这些条件的所有规则，你的服务将一直陷入正在激活状态(Activating)。服务将不停为容器查找可用的主机。你能编辑服务的规模活着添加/编辑另一台满足这些规则的主机。
- 应该或不应该。Rancher试图使用匹配这些字段和值的主机。在没有主机匹配所有规则的条件下，Rancher将一个接一个移除软限制直到有主机满足剩下的限制。

字段

- 主机标签：当为容器/服务选择主机时，Rancher将会检测主机上的标签来查看是否这些标签匹配所提供的键/值对。因为每台主机能有一个或多个标签，Rancher将把键/值对和主机上所有标签做对比。当添加一台主机到Ranche时，你就能为主机添加标签。你也能通过主机下拉菜单中的**Edit**选项来编辑主机上的标签。激活主机上的标签列表可从下拉菜单的键值字段获得。
- 带标签的容器：当选择这个字段时，Rancher将查找主机上存在已经有标签匹配键值对的容器的主机。因为每个容器能有一个或多个标签，Rancher将把键值对和主机上的每个容器的所有标签进行比较。容器标签在容器的**Advanced Options -> Lables**选项卡中。你不能在容器启动后编辑容器的标签。为了能创建拥有同样设定的新容器，你能在容器启动前**Clone**容器/服务以及增加新的标签。正在运行的容器的用户标签列表看从下拉菜单的键值字段获得 带名字的服务：Rancher将检测一台主机上是否有指定服务的容器在运行。如果在稍后的时间，服务改变了名字或者取消处于未激活或者已移除状态，这规则就不再有效。如果你选择了这个字段，输入的值必须为 `栈名/服务名 (stack_name/service_name)` 的格式。正在运行的服务的列表可以从下拉菜单中的值(value)字段获得。
- 带名字的容器：Rancher将检测一台主机上是否指定的名字的容器正咋运行。如果稍后的时间，容器改变了名字或者处于未激活/已移除状态，整条规则变不再有效。运行中的容器列表可以从下拉菜单的值（value)字段获得。

服务/堆栈

增加服务

增加服务别名

增加负载均衡

增加外部服务

堆栈选项

基础架构/主机

从主机开始

从主机开始

在 Rancher 中，我们在UI上提供了很简单的指导，来将那些直接支持的云提供商所提供的主机添加进来，同时也提供了指导，在你的云提供商不支持的时候,添加你拥有的主机。在 **Infrastructure** 中的 **Hosts** 页，点击 **Add Host** 。

主机要求

- 任何现代的 Linux 发行版，支持 Docker 1.10.3。RancherOS, Ubuntu, RHEL/CentOS 7 经过了更严格的测试。
- 1GB RAM
- 建议 CPU w/ AES-NI

主机如何工作?

在 rancher 代理端在主机上启动后，主机就连上了 Rancher 服务端。注册令牌，就是 **Add Host -> Custom** 页面中的长URL，用来帮助 rancher 代理端首次连上服务端。在连接中，将会在 Rancher 服务端生成一个代理端帐号和 API 密钥对。密钥对用来作为后来的所有通讯，使用如其他类型的帐号如环境 API 密钥，相同的认证与授权逻辑。

设计如此是因为代理端是不可信的，因为他运行在外部的和（对服务端）有潜在敌意的硬件上。代理端帐号仅能在 API 中访问他们所需的资源，对事件的回复会被检查确保事件确实发送给此代理端等。由于没有反方向的为代理端来验证主机，因此你可以设置 TLS 并且证书将会被验证。

IPSec 密钥是每个 环境独有的。他由服务端生成，存储在数据库中，并作为代理端注册 API 密钥对的一部分发送给主机。连接是在主机间点对点并 AES 加密的，其中加密能被大多数的现代 CPU 加速。

添加主机

你第一次添加主机时，你可能被要求设置好 主机注册。设置将检测何种DNS名称或ip地址及端口用来使你的主机链接到 Rancher API。默认的情况下，我们选择管理服务端 IP 和端口 `8080` 。 如果你选择改变地址，请确保指定端口用来连接到 Rancher API。任何时候你都能够更新 主机注册。在设置好你的主机注册后，点击 **Save**。

我们支持直接从云提供商添加主机，或添加已经被配置好了的主机。对于云提供商，我们使用 `docker-machine` 进行配置，并支持任何 `docker-machine` 支持的镜像。

选择你想添加的何种主机类型：

- 添加自定义主机
- 添加 Amazon EC2 主机
- 添加 Azure 主机
- 添加 DigitalOcean 主机
- 添加 Exoscale 主机
- 添加 Packet 主机
- 添加 Rackspace 主机
- 添加其他驱动的主机

当一台主机加入 Rancher，rancher 代理端将会在主机上启动。Rancher 将会自动拉取 `rancher/agent` 的正确镜像版本标签并运行所需的版本。对于每个 Rancher 服务端版本，代理端版本都被明确标记。

主机标签

对于每个主机，你能添加标签来协助组织你的主机。当启动 rancher/agent 容器时，标签作为环境变量被添加。UI中的主机标签将会是 key/value 对并且key都需要是唯一标识。如果你添加了两个相同的 key 带有不同 value，我们将使用后输入的 value 作为 key/value 对。

通过对主机添加标签，你能在调度服务/负载均衡/服务中使用标签，创建对你的服务运行的主机白名单或黑名单。

如果你计划使用 外部 DNS 服务，并需要 编辑 DNS 记录来使用非主机IP的其他IP，你需要在主机上包含此标签 `io.rancher.host.external_dns_ip=<IP_TO_BE_USED_FOR_EXTERNAL_DNS>` 。主机标签能在注册主机或在主机添加进 Rancher 后添加，但是应该在外部 DNS 服务开启前添加到主机。标签的值将会在编辑外部 DNS 服务的规则时被使用。

当使用UI添加不同云提供商的主机时，rancher/agent 命令会自动为你执行并带有通过UI添加的主机标签。

当添加自定义主机，你能通过UI添加标签并且它将自动添加带有 key/value 对的环境变量 (`CATTLE_HOST_LABELS`)到UI界面的命令中。

例子

```
# 添加一个主机标签到 rancher/agent 命令中
$  sudo docker run -e CATTLE_HOST_LABELS='foo=bar' -d --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock rancher/agent:v0.8.2 \
    http://<rancher-server-ip>:8080/v1/projects/1a5/scripts/<registrationToken>

# 添加多于一个主机标签需要使用 `&` 连接多余的主机标签
$  sudo docker run -e CATTLE_HOST_LABELS='foo=bar&hello=world' -d --privileged \
    -v /var/run/docker.sock:/var/run/docker.sock rancher/agent:v0.8.2 \
    http://<rancher-server-ip>:8080/v1/projects/1a5/scripts/<registrationToken>
```

> 注意： `rancher/agent` 的版本与 Rancher 服务端的版本相关联。你需要检查自定义命令以便获取所使用版本的正确标识。

自动实现的主机标签

Rancher 自动创建关于 Linux 内核版本和 Docker Engine 版本的主机标签。

Key	Value	描述
`io.rancher.host.linux_kernel_version`	主机的 Linux 内核版本 (例如： `3.19`)	主机上运行的 Linux 内核的版本
`io.rancher.host.docker_version`	主机的 Docker 版本 (例如： `1.10.3`)	主机上的 Docker Engine 版本

代理后的主机

为了支持代理后面的主机，你需要编辑 Docker daemon 来指向代理。详细指导列在我们的 添加自定义主机 页面

访问来自于云提供商的主机

在 Rancher 启动主机后，你可能需要访问主机。我们提供了所有在启动设备时生成的证书，可通过下载方式获取。在主机的下拉菜单中点击 **Machine Config**。那将会下载一个包含所有证书的 tar.gz 文件。

为了 SSH 进入你的主机，进入你的终端/命令行。进入所有证书存放的文件夹，然后通过使用 `id_rsa` 证书以便 ssh 进入主机。

```
$ ssh -i id_rsa root@<IP_OF_HOST>
```

克隆主机

由于启动云提供商上的主机需要使用 access key，你可能想要更简单的创建另一个主机而不需要再次输入所有的证书。Rancher 提供了通过克隆这些证书来启动一个新主机的功能。从主机的下拉菜单中选择 **Clone**。那将会引向 **Add Host** 页面并带有克隆主机的所使用的证书。

编辑主机

所有能对主机做的操作选项都位于主机的下拉菜单中。在 **Infrastructure -> Hosts** 页面中，下拉标识将会在你鼠标放置在主机上时出现。如果你点击主机名字来查看主机的更多信息，下拉标识将会出现在页面的右上角。它将会位于主机状态的旁边。

如果你选择 **Edit**，你能更新名字，描述或主机的标签。

停用/激活主机

停用主机将使主机进入 *Inactive* 状态。在此状态中，将不会部署新的容器。任何此主机上的活动的容器将会保持激活，并且你仍然能够对这些容器执行操作（开始/停止/重启）。主机将仍然连接到 Rancher 服务端。从主机的的下拉菜单中选择 **Deactivate**。

当主机处于 *Inactive* 状态，你能通过点击主机下拉菜单中的 **Activate** 使主机重回 *Active* 状态。

> 注意： 如何在 Rancher 中一台主机失效 (例如 处于 `reconnecting` 或 `inactive` 状态)，你将需要执行 健康检查 以便 Rancher 在其他不同的主机上启动你服务中的容器。

删除主机

为了从服务端删除主机，你需要在下拉菜单中操作一系列的步骤。

选择 **Deactivate**。当主机完成停用，主机将显示 *Inactive* 状态。选择 **Delete**。服务端将开始将主机从 Rancher 服务端移除的过程。在它完成删除后首先显示的状态将会是 *Removed*。在从UI中立即消失前，将会继续以便完成移除过程，并进入 *Purged* 状态。

如果使用 Rancher 在云提供商上创建的主机，主机将会在云提供商处删除。如果主机是通过使用 自定义命令添加的，主机将会保留在云提供商处。

> 注意： 对于自定义主机，所有容器，包括 Rancher 代理端将会继续保留在主机上。

删除 Rancher 外的主机

如果你的主机在 Rancher 外被删除了，Rancher 服务器将会继续展示此主机直到它被移除。最终，这些主机将显示为处于 *Reconnecting* 状态但无法重新连接。你能够 **Delete** 这些主机用于从UI上移除他们。

添加自定义主机

Amazon EC2

Azure

Digital Ocean

Exoscale

Packet

Rackspace

其它驱动

基础架构/容器

基础架构/证书

Rancher目录

为了轻松的部署复杂的Stacks（应用栈），Rancher为我们设计了称为Catalog（应用程序模板“目录”）的功能。访问**Catalog**页, 将会看到有两个子页：ALL以及LIBRARY。ALL是所有已启用的应用程序模板集（包括LIBRARY下的项目和自建项），而**Library**页下面包含了Rancher certified catalog和community-catalog。前者是Rancher Labs公司官方认证的项目，后者是由社区维护的项目。Rancher Labs公司仅仅只对通过认证的项目进行维护与支持。两者的不同点可以从项目缩略图顶部是否带有“Certified”进行区分。

Rancher系统的admin角色可以对目录下的项目进行添加或删除操作，在**Admin -> Settings**页我们可以选择在系统里显示（启用）哪些目录项。添加一个目录项并不复杂，只需要给定一个名字和一个URL。对于URL的内容要求是它必须能被 `git clone` 处理。当添加一个目录项后，它会立即显示在目录页并处于可操作状态。

URL示例：https://github.com/rancher/rancher-catalog.git （下面会提到如何准备正确的文件及目录结构并提交至git服务器）

如果Rancher Server处于http proxy代理服务器之后，那么需要修改Docker守护进程的启动参数，否则catalog将无法正常运作。

启动模板

首先找到需要的模板，可以进行关键字搜索或直接进行类型过滤。找到模板之后就可以点击View Details按钮进去，填写必须的信息之后就可以点**Launch**按钮启用模板了。

1. 对于模板项的版本，系统默认采用的是当前最新的版本，如果需要的话，仍然可以选择存在的其它老版本。
2. 给定一个新的**stack**名字，必要时填写stack的描述信息。
3. 选择或填写**Configuration Options**里的项目, 这些项目因不同的目录特性而不同，如果勾选**Start Services after creating**，那么一旦点击**Launch**，服务将会自动启动。
4. 点击**Launch**以创建基于模板的应用栈(stack)。点击**Preview**则可以以展开方式查阅 `docker-compose.yml` 和 `rancher-compose.yml` 等文件内容，这些正是用以创建应用栈(stacks)的模板文件。

升级模板

Rancher里有个很棒的功能是当新版本的模板上载到目录之后，它会提示你有新版本可用。当点击**Upgrade Available**时，当选择了某个版本之后，需要查阅一下**Configuration Options**是否也需要做出相应的改变，确认无误之后即可点击**Save**保存变更。

当所有的服务被升级之后，服务service和应用栈stack的状态会显示为**Upgraded**。如果对升级的结果表示满意，那么最后就可以点击stack下拉菜单里的**Finish Upgrade**以确认升级。注意：一旦确认完成升级过程，就没法再回退到旧的版本了。

版本回退

如果在升级过程中遇到错误，需要回退至升级前的正常状态，可以在stack的下拉菜单里选择**Rollback**回退至之前的版本。

创建私有目录

Rancher catalog 服务定义了一套配置文件的格式，只需按照格式来添加配置文件，catalog服务就能通过git自动把它加到rancher-server的容器的模板目录下。

目录结构

目录模板在Rancher里的显示内容基于环境的集群管理类型。

- *Cattle* 环境: UI里的条目来自 `templates` 文件夹
- *Kubernetes* 环境: UI里的条目来自 `kubernetes-templates` 文件夹
- *Swarm* 环境: UI里的条目来自 `swarm-templates` 文件夹

```
-- templates OR kubernetes-templates OR swarm-templates
  |-- cloudflare
  |   |-- 0
  |   |   |-- docker-compose.yml
  |   |   |-- rancher-compose.yml
  |   |-- 1
  |   |   |-- docker-compose.yml
  |   |   |-- rancher-compose.yml
  |   |-- catalogIcon-cloudflare.svg
  |   |-- config.yml
...
```

在主目录里，需要有一个 `templates` 文件夹， `templates` 文件夹里包含了已创建好的每个catalog条目。我们推荐对每一个catalog条目使用一个简单的模板名，并和文件夹名称保持一致。

在catalog条目文件内（例如 `cloudflare` ）对应于创建的每一个不同版本都有一个对应的文件夹。第一个版本对应的文件夹名是 `0` ，后续版本则依次增加，例如第二个版本对应于文件夹 `1` 。提供了一个新版本对应的文件夹内容，就提供了对stack升级其模板至新版本的渠道。作为一种可选方案，还可以直接对 `0` 文件夹也就是第一版本文件夹内容直接更新然后再重新部署这个条目

> 注意：每个catalog条目必须使用独立单词，所以请在较长的完整名称中使用连字符 `-` 代替空格。而在 `config.yml` 的 `name` 部分内你可以使用空格。

Rancher目录里显示的目录文件

在catalog条目文件夹里有两个文件，其细节决定了条目如何在Rancher Catalog内具体显示。

- 第一个配置文件 `config.yml` 包含了条目细节。

```
name: # Name of the Catalog Entry
description: |
   # Description of the Catalog Entry
version: # Version of the Catalog to be used
category: # Category to be used for searching catalog entries
maintainer: # The maintainer of the catalog entry
license: # The license
projectURL: # A URL related to the catalog entry
```

- 第二个文件是catalog条目的图标文件，其文件名必须以 `catalogIcon-` 作为开头。

对于每一个catalog条目文件夹，至少需要三项，其中包含两个文件和一个文件夹，它们是 `config.yml` ， `catalogIcon-entry.svg` 和 `0` 文件夹（第一个版本），如果有多个版本，则包含更多的文件及文件夹。

Rancher 目录模板

`docker-compose.yml` 和 `rancher-compose.yml` 在Rancher里是必需的文件，被rancher-compose工具调用以启动服务，它们位于版本号对应的文件夹内（例如 `0` ， `1` 等）。

`docker-compose.yml` 同时应该是一个能够被 `docker-compose` 工具载入并启动的文件，其格式遵循docker-compose规范。

`rancher-compose.yml` 则包含了附加的信息用于自定义catalog条目。在 `.catalog` 节，有一些字段是为了正确执行catalog条目所必需的项目。

`README.md` 作为可选项也可能被创建，其主要作用在于提供详尽的描述说明或注意事项。

`rancher-compose.yml`

```
.catalog
  name: # Name of the versioned template of the Catalog Entry
  version: # Version of the versioned template of the Catalog Entry
  description: # Description of the versioned template of the Catalog Entry
  uuid: # Unique identifier to be used for upgrades. Please see note.
  minimum_rancher_version: # The minimum version of Rancher that supports the template
  questions: #Used to request user input for configuration options
```

> 注意： `uuid` 是一个必选参数用于升级更新版本的场景。每个 `uuid` 都必须是唯一的并且随着版本变化而增加。推荐的格式是使用catalog条目名称并添加一个版本号前缀 `-0` ，如前所述对版本所做的说明，下一个版本前缀则是 `-1` 。

`rancher-compose.yml` 内的Questions

位于 `.catalog` 里的 `questions` 小节允许用户修改服务的配置选项。而文件 `answers.txt` 则对应地设置这些配置项的具体值。

每一个配置项都在rancher-compose.yml的 `questions` 小节内列出。

```
.catalog
  questions:
    - variable: # A single word that is used to pair the question and answer.
      label: # The "question" to be answered.
      description: | # The description of the question to show the user how to answer
the question.
      default: # (Optional) A default value that will be pre-populated into the UI
      required: # (Optional) Whether or not an answer is required. By default, it's co
nsidered `false`.
      type: # How the questions are formatted and types of response expected
```

Type

`type` 小节用于服务配置项的问题及对应答案以何种方式显示于UI界面。

合适的格式有：

`string`

显示一个文本框，用于存放字符串型的在 `answer.txt` 里定义的值。

`int`

显示一个文本框，用于存放被整形化的在 `answer.txt` 里定义的值。UI会对其数值在模板启用前做合法性检测。

`boolean`

显示一个单选按钮，用于存放其在 `answer.txt` 里定义的布尔变量值，如果布尔值为 `真` ，则单选框被勾选，否则被置空。

`password`

显示一个文本框，用于存放其在answer.txt里定义的密码字符串内容。

`service`

显示一个下拉框，其内容为环境(environment)里的所有服务(services)。

`enum`

显示一个下拉框，其内容为自定义的 `options` 条目。

```
.catalog
  questions:
    - variable:
      label:
      description: |
      type: enum
      options: # List of options if using type of `enum`
        - Option 1
        - Option 2
```

`multiline`

显示一个多行文本框。

```
.catalog
  questions:
    - variable:
      label:
      description: |
      type: multiline
      default: |
        Each line
        would be shown
        on a separate
        line.
```

Kubernetes

英文原版与中文翻译版初稿 by Jordan

To deploy Kubernetes in Rancher, you’ll first need to create a new environment that has specified the cluster management to be Kubernetes. 为了在Rancher上部署Kubernetes，首先你需要创建一个指定用Kubernetes管理的集群环境。

CREATING A KUBERNETES ENVIRONMENT 创建一个KUBERNETES环境

In the dropdown of environments, click on the Manage Environments. To create a new environment, click on Add Environment, select Kubernetes as the cluster management, provide a Name, Description (Optional). If access control is turned on, you can add members and select their membership role. Anyone added to the membership list would have access to your environment. 在下拉式环境中，点击管理环境。 为了创建一个新的环境，点击“Add Environment”，选择Kubernetes做为集群管理者，起一个名字，Description (可选项)。 如果访问控制权限是开着的，你可以增加成员，选择membership角色。 在membership列表中的任何成员都可以访问你的环境。

After a Kubernetes environment has been created, you can navigate to the environment by either selecting the name of the environment in the environment’s dropdown in the upper right hand corner or by selecting Switch to this Environment in the specific environment’s drop down. 在Kubernetes环境创建后，你既可以通过选择环境名字又可以通过切换环境在下拉菜单中，浏览你的环境。

NOTE: As Rancher adds support for multiple cluster management frameworks, Rancher currently does not support the ability to switch between environments that already have services running in it. 注意： Rancher支持多种集群管理框架，目前不支持已经运行服务的环境之间切换。

STARTING KUBERNETES 启动KUBERNETES After a Kubernetes environment has been created, you can start the Kubernetes cluster by adding at least one host to your environment. The process of adding hosts is the same steps for all cluster management types. Once the first host has been added, Rancher will automatically start the deployment of the required Kubernetes components (i.e. master, kubelet, etcd, proxy, etc.). You can see the progress of the deployment by accessing the Kubernetes tab. 在Kubernetes环境创建后，你可以启动Kubernetes集群通过至少增加一个主机到你的环境。在所以集群管理类型中

添加主机的流程是一样的步骤。 一旦第一个主机被添加后，Rancher将自动开始部署Kubernetes组件（如，master, kubelet, etcd, proxy等）。你可以通过Kubernetes选项卡看到部署过程。

USING KUBERNETES 使用KUBERNETES Once the setup has completed, you can begin to create or manage your own Kubernetes applications via the following ways: 一旦配置完成，你就可以通过下面方法开始创建或管理你自己的Kubernetes应用。

RANCHER UI RANCHER 用户界面 Rancher provides full CRUD capability of creating services, replication controllers (RCs), and pods. In the Kubernetes tab, click on the one of these items and click Add. A kubernetes template will be shown in the UI and is editable. After you have made changes to the configuration file, click on Create. Rancher对创建services, replication controllers (RCs)和pods提供了增删改查的能力。 在Kubernetes tab中点击这些项目和点击添加，一个可编辑的kubernetes模板将会被展现出来，然后你在配置文件修做一些改，点击创建。

RANCHER CATALOG RANCHER 服务目录 Rancher supports the capability of hosting a catalog of Kubernetes templates. To use a template, click on the Catalog tab. Select the template that you want to launch and click View Details. Review and edit the stack name, stack description, and configuration options and click on Launch. Rancher支持Kubernetes模板的服务目录的能力。要使用模板，请单击“服务目录”选项卡。选择你要启动的模板，然后单击“View Details”。审查和编辑堆栈名称、堆栈描述和配置选项，然后单击“启动”

If you want to add your own templates to Kubernetes, you add them to the Rancher catalog and place your templates in a kubernetes-templates folder. 如果你想添加自己的模板到Kubernetes，你将它们添加到Rancher的服务目录，把你的模板放在Kubernetes模板文件夹里。

KUBECTL KUBECTL To configure your own kubectl to talk to your newly created Kubernetes cluster, go to Kubernetes -> kubectl -> Generate Config to generate the necessary kube/config_file that you can download and add to your local directory. 为了配置你自己的kubectl与新创建的Kubernetes集群关联，通过Kubernetes -> kubectl -> Generate Config 生成所需的kube/config_file，你可以下载并添加到你的本地目录。

KUBECTL VIA SHELL KUBECTL VIA SHELL Rancher provides a convenient shell access to a managed kubectl instance that can be used to manage Kubernetes clusters and applications. Rancher提供了一个方便的Shell进入到托管kubectl实例来管理Kubernetes集群和应用。

KUBERNETES NAMESPACES KUBERNETES命名空间 Rancher supports the ability to manage different Kubernetes namespaces. In the upper right hand corner, you will be able to see which Namespace that you are working in. After the first host is added, Rancher

creates the default namespace. Rancher提供了支持管理不同的Kubernetes命名空间的能力。在右上角，你可以看到你在工作中的那个名字空间，在添加第一主机后，Rancher创建默认的命名空间。

ADDING NAMESPACES 添加NAMESPACES To add an additional namespace into Kubernetes, click on the current namespace and a dropdown of available namespaces and Manage Namespaces will appear. Click on Manage Namespaces. 为了添加一个额外的命名空间到Kubernetes，点击当前命名空间和下拉可用的命名空间，“Manage Namespaces将会出现。单击”Manage Namespaces“。

In the Namespaces page, click on Add Namespace. Update the configuration file and click Create. 在命名空间中的页面中点击“Add Namespace”，更新配置文件，然后单击“创建”。

EDITING NAMESPACES 编辑NAMESPACES For existing namespaces, in the Namespaces page, click on Edit in the namespace's dropdown to update it. Click on Save. 对于现有的命名空间，在命名空间的页面中点击编辑命名空间的下拉菜单，然后更新，点击保存。

SWITCHING NAMESPACES 切换NAMESPACES In the dropdown of namespaces, you can select the namespace that you want to launch services in to switch between the namespaces. 在命名空间的下拉菜单，你可以选择你想启动服务的命名空间，在名称空间之间进行切换。

Swarm

若想在Rancher中使用Swarm，首先需要创建一个集群管理方式配置为**Swarm**的环境。

创建Swarm集群管理的环境

在右上角的环境下拉菜单中，单击管理环境。要创建一个新环境，单击新建环境，选择**Swarm**作为集群管理方式，提供名称，说明（可选）。如果打开了访问控制功能，你可以添加成员并赋予相应的角色。所有位于成员列表的用户都可以访问此新增环境。

创建完由Swarm管理的环境后，你可以通过在右上角的环境下拉框中根据环境名字切换到对应的环境。

> 注意 Rancher目前支持多种集群管理方式，但是不支持在已有服务运行的环境中切换集群管理方式。

启用Swarm

创建完由Swarm集群管理的环境后，现在可以添加至少一台主机来启用Swarm集群。对于不同的集群管理方式，添加主机的过程都是一样的。一旦添加完一台主机，Rancher就会自动部署Swarm需要的组件（比如swarm和swarm-agent）在主机上。你可以在Swarm标签页看到部署的进度。

> 注意 swarm不需要安装在所有的主机上。

使用Swarm

swarm组件部署完成后，你就可以通过以下方式来创建或者管理Swarm程序：

RANCHER界面

用户可以在Rancher界面上完成创建、获取、更新、销毁项目的所有操作。在Swarm标签页，选择项目并单击创建项目。创建项目时，你可以在界面上通过上传文件或者粘贴文件内容的方式来提供docker-compose.yml。如果你的项目装配模板中包含环境变量，你需要通过添加变量替换的方式来声明变量，最后单击创建。

RANCHER目录

Rancher支持加载整套Swarm模板。如果使用模板，可以单击目录标签页。选择你想加载的模板并单击查看详情。审核并编辑工作站名称，工作站说明和相关的配置，然后单击加载。 如果你想添加模板到Swarm，你可以先添加到Rancher目录中，并放置在swarm-templates目录。

命令行界面

为了配置你的工作机器用来管理swarm，你可以依次单击**Swarm->CLI->**生成配置来生成必要的API秘钥并将配置文件放置在docker-cli.zip压缩文件中。遵循界面上的说明去配置TLS并连接到Docker。

SHELL脚本

Rancher提供了一套方便的SHELL接口来执行docker或者docker-compose命令。

使用标签

升级 Rancher

如果你运行Rancher服务端时 没有 使用 外部数据库, Rancher服务端数据库在你的Rancher服务端容器内. 我们将要使用运行中的Rancher服务端容器来创建一个数据容器. 数据容器将会通过使用 `--volumes-from` 以用来启动新的Rancher服务端. 或者, 你能将数据库从容器中拷贝出来到主机的某个目录然后绑定挂载数据库.

> 注意: 如果你使用了外部数据库, 你能停止原始的Rancher服务端容器然后使用相同的外部数据库指引来启动新版本的Rancher服务端.在新服务端启动运行后,你就能删除老的Rancher服务端容器了. 注意: 如果你仅停止容器,如果你的机器重启,容器将会由于有 `--restart=always` 而重新运行.

通过创建数据容器升级Rancher

1. 停止容器.

```
$ docker stop <container_name_of_original_server>
```

2. 创建一个 `rancher-data` 容器. 注意: 此步骤可以跳过,如果你过去已经升级过并且已经有了一个 `rancher-data` 容器.

```
$ docker create --volumes-from <container_name_of_original_server> \
 --name rancher-data rancher/server:<tag_of_previous_rancher_server>
```

3. 拉取最近的Rancher服务端镜像. 注意: 如果你跳过此步骤然后尝试运行 `最近的` 镜像, 将不会自动拉取更新后的镜像.

```
$ docker pull rancher/server:latest
```

4. 启动一个新的Rancher服务端并使用来自 `rancher-data` 容器的数据库. Rancher内的任何修改都将会保存到 `rancher-data` 容器.如果你在服务端件到了关于日志锁的异常,请参考如何处理日志锁.

> 注意: 取决于你运行了Rancher服务端多久, 某些数据库迁移可能花费比预计更长的时间.请不要在升级中间时停止升级,否则在下次升级时将会遇上数据库迁移错误

```
$ docker run -d --volumes-from rancher-data --restart=always \
  -p 8080:8080 rancher/server:latest
```

> 注意: 如果你在原始的Rancher服务端安装时设置过任何环境参数或传入过ldap认证, 你将需要在命令中加入这些环境变量或认证.

5. 移除旧的Rancher服务端. 注意: 如果你仅停止了容器,容器将会由于有 `--restart` 而在你的机器重启后重新运行. 我们建议在你的升级成功后移除此容器.

通过启动时使用绑定挂载升级Rancher

1. 停止运行中的Rancher服务端容器.

```
$ docker stop <container_name_of_original_server>
```

2. 从服务端容器中拷贝数据库文件出来. 注意:如果你已经有数据库存储在主机上了,你能够跳过此步骤.另外, 如果数据库已经被拷贝到了容器外面,它将会在 //mysql/ 里面,因为Docker的拷贝方法如此. 在绑定挂载到容器里时一定要考虑到这一点. 如果你开始绑定挂载,你将不需要输入 mysql/.

```
$ docker cp <container_name_of_original_server>:/var/lib/mysql <path on host>
```

3. 现在为文件夹设置 UID/GID 以便容器内的mysql用户对mysql挂载有正确的所有权.

```
$ sudo chown -R 102:105 <path on host>
```

4. 开启新的服务端容器.

```
$ docker run -d -v <path_on_host>:/var/lib/mysql -p 8080:8080 \
  --restart=always rancher/server:latest
```

> 注意: 如果你是从前一个容器中拷贝出来的数据库,确保在主机路径最后是 '/' 将很重要.否则目录会在错误的地方结束.

Rancher代理端

每个Rancher代理端版本都固定了相应的Rancher服务端版本.如果你升级了Rancher服务端并且Rancher代理端需要升级,它将会自动升级代理端到最新版本的Rancher代理端.

对于使用 `rancher/agent-instance` 镜像的任何东西,运行中的容器将获得升级,甚至是容器的镜像没有升级到最近版本.对于 *Network Agent*, 这将在有关网络的触发器触发时进行(例如, 添加另一个容器,删除一个容器).对于 *Load Balancers*,升级将在下一次负载均衡配置更新时进行(例如,目标实例重启,新的目标服务加入,主机名路由规则变更,负载均衡实例重启等).

没有互联网接入的用户

没有互联网的用户需要将最新的 `rancher/agent-instance` 下载到他们自己的镜像库. 为了升级网络代理的版本,你需要通过UI手工停止网络代理端并移除他们. 在缺失网络代理端的主机上, 任何触发网络的事件都将会引起一个新的网络代理端(最新版本)启动. 为了升级你的负载均衡的版本, 你将需要重新创建它们以便获取到最新版本的 `rancher/agent-instance` .

为 Rancher 做贡献

参与开发

查看GitHub 代码库, 在我们的 wiki 文档里有给 Rancher 开发人员写的如何上手开发工作的文档。

先从 cowpoke 来开始您的代码协作开发之旅吧！

代码库

在我们的 GitHub 首页 您能看到 Rancher 所有的相关代码库，但是下面我们会简要介绍 Rancher 所使用的几个核心的代码库。

Rancher Repo: 这个代码库的功能是把所有其它相关代码库集成在一起。

Cattle Repo: 这是用来开发 Rancher 核心功能的代码库。

UI Repo: 这是用来开发 Rancher 图形用户界面部分的代码库。

Community Catalog Repo: 这里是 Rancher Catalog 所使用的所有社区贡献的模板。我们欢迎您在这里贡献模板。

Rancher-Compose Repo: 这是 rancher-compose CLI 功能的代码库。它总是会与 docker/libcompose 保持同步。

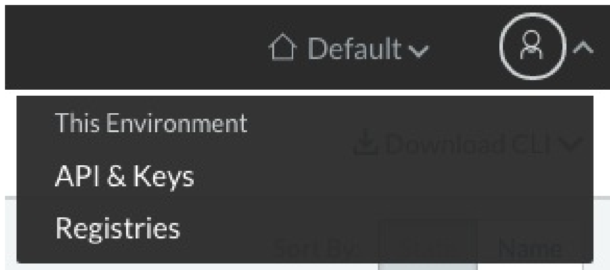
关于 Bug

如果您发下了任何 Bug 或者有任何使用问题，请通过填写 Issue 的方式联系我们。尽管我们的 Rancher 中有很多代码库，我们还是想让大多数的 bug 都能被记录在 Rancher repo ，这样就不会被错过！

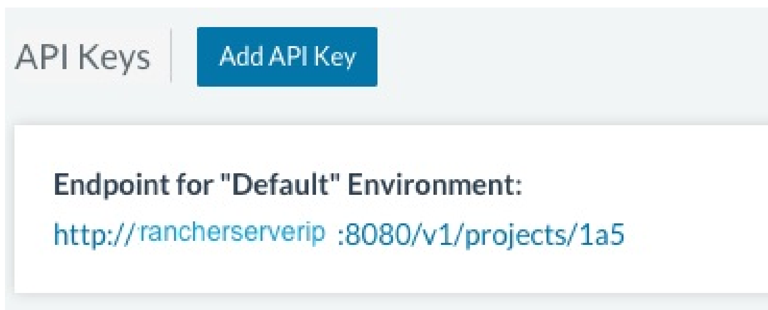
如果您想帮助优化我们的文档，请提交 PR 到我们文档的代码库 docs repo ，或者点击文档页面上的 **Edit this page** 按钮，它会直接带您进入页面修订功能。

API实践

API有可以从web浏览器访问的用户接口。这是查看资源，执行动作，以及查看等价于cURL命令执行的HTTP请求&回应。点击**API**来获得URL endpoint来访问API。



点击端点链接：



术语

API中一些资源类型命名和当前UI上所用的术语，并不一致。尤其是：

UI	API	Description
Environment	project	一组物理资源， 例如 主机
Stack	environment	一个(API) 环境是一组服务以及rancher-compose所操作的级别。

在这文档中，我们使用UI所用的术语进行描述，在不同点也会提供额外的声明。这些混乱将会在未来的 `/v2` 版本API清除。

验证

如果Access Control启动的话，API请求必须包含验证信息。验证操作通过带有API keys的HTTP基本验证来完成。API密钥或者属于一个(UI) Environment/ (API) Project，只能访问该Environment；或者属于一个Account，能够访问该账户所有的Environment，也能够创建新的Api密钥。在UI上有单独的JSON Web Token接口。

一个环境的API密钥

环境的API密钥能够在UI中创建，查看[API & Kes[(/configuration/api-keys/)。 密钥所属于一个环境，而且完全有权管理该环境，但无权管理其他环境。Membership roles并不支持密钥。

账户的API密钥

账户的API密钥当前并没有在UI中显示。你可以点击浏览器的API页面来创建一个:

- 点击UI上的Endpoint链接
- 导航到 `/v1/apikeys`
- 点击 `Create`
- 选择你要创建密钥的 `accountId`
- 可选择设置 `name` 和 `description`
- 点击 `Show Request` ， 然后 `Send Request`
- 保存回应中的 `publicValue` 和 `secretValue`

Account密钥能够创建新的Environments，也能够用来通过 `/v1/projects` 来访问多个Environments。这些密钥所采用的Membership roles限制该Account哪些Environments以及哪些动作能够使用。

作用域

- 大部分的资源被一个Environment所拥有，并通过 `/v1/projects/<project_id>/<resources>` 地址进行访问。
- 因为环境证书只能够存取一个Environment(project)，你可以选择跳过 `/project/<project_id>` 部分。 -例如，如果你在project `1a5` 使用了一个Environment API密钥，那么 `/v1/projects/1a5` 等同于 `v1` ， `/v1/projects/1a5/hosts` 等同于 `/v1/hosts` 。
- 这篇文档一般只涉及更短的 `/v1/<type>` 版本.如果使用了Account密钥，则将适当的environment添加到路径中。

发送请求

这些API一般都是RESTful API，但拥有一些特性使得客户端能够查看任何资源的定义，因而可以编写通用的客户端，而不是每种资源类型编写特定代码的客户端。对那些想费力深入通用API规范细节的人，查看这里

- 每种类型有一个Schema，负责描述：
  - 到达该资源类型的collection的URL
  - 该资源拥有的所有域，连同域的类型，域的基本验证规则，它们是必须还是可选等.
  - 该类型资源能够执行每个动作，以及输入和输出(和schemas一样)
  - 每个域允许的过滤方式
  - collection中哪些HTTP动词方法对其或者其中独立的资源可用
- 因此理论上你能够只需要加载schemas列表就知道API的一切。这就是实际上UI如何工作的，它并没有包含和Rancher特定的代码。获取Schemas的URL，作为一个 `X-Api-Schemas` 头部，在每一个HTTP回应被发送。从该URL，你可以通过访问每一个schema的 `collection` 链接，来知道哪里显示资源；通过返回资源的其他 `links` 来获取其他信息。

- 实际上，你可能只想构建URL字符串。我们强烈建议限制构建的URL字符串只在顶级显示collection(`/v1/<type>`)或者获取一个特定的资源(`/v1/<type>/<id>`)时使用。任何更底层的动作都将在将来版本改变。(译者注：这里指用户应尽通过可能返回的HTTP回应信息中提取想要资源的URL，不要根据当前版本资源的URL结构自己构建。)

- 资源间的关系称为links。每个资源都包含一个 `links` 表，其中包含link的名字以及获取link信息的URL. 再次强调，你应该通过发送 `GET` 请求来获取资源以及使用 `links` 表中的URL，而不是自己构建这些字符串。

- 大多数资源都有些动作，用来处理某事或者改变资源的状态. 通过发送 `POST` 请求到 `action` 表中的URL来执行你想要的动作. 一些动作会需要输入或者会产生输出，请查看每种类型资源或者schemas的独立文档来获取特定信息.

- 编辑资源， 则发送一个带有资源的域数据的 `PUT` HTTP请求到你想要改变的资源的 `links.self` 链接.未知域或者不可编辑的域将被忽略.

- 删除资源， 则发送一个 `DELETE` HTTP请求到资源的 `links.self` 链接. 注意一些资源必须要在删除之前停止;被删除的资源仍然通过API根据指定资源ID来恢复.

- 创建资源，发送一个 `POST` HTTP请求schema中的collection URL(也即是 `/v1/<type>`).

过滤

大多数的collections能够在服务端通过HTTP的检查参数按照通用域进行过滤. `filter` 表显示哪些域能够被过滤， 以及哪些值是你的请求过滤的. API UI控制了安装过滤的过程，为你显示正确的请求。"equals"和 `field=value` 相匹配。修饰符可以添加到域名，例如， `field_gt=42` 意思就是 `field大于42` 。更多细节查看API spec。

排序

大多数的collections在服务端通过HTTP的查询参数按照通用域进行排序. `sortLinks` 表将显示哪些排序是可用的，连同显示已经按照该方式排序的collection的URL.如果指定的话，它也将包含当前回应是以何种方式排序的信息。

分页

API回应默认以每页100资源的限制进行分页.可以通过添加 `limit=1000` 查询参数来将上限提升到1000. collection回应中的 `pagination` 会告诉你是否收到全部结果；如果没有收到全部结果，将会有一个链接指向下一页。

WebSockets

一些Rancher特性，比如容器日志，shell访问，统计都使用websockets来传输信息流。通过API来使用：

- 点击适当的链接或者执行适当的动作
- 回应将包含一个URL(以 `ws://` 或者 `wss://` 开头)以及一段长的 `token` 字符串.
- 打开一个指向返回URL的WebSocket客户端
- token是由Rancher server签名的，允许所在的主机容器授权这次请求，所以它作为一个HTTP的头部，如 `Authorization: Bearer <token_string>` ，发送给Rancher server。
- 如果你通过web浏览器发送请求，你不能发送包容任意HTTP头部的请求，因此你可以选择的是将token添加到URL中作为URL查询参数来代替，如 `token=<token_string>` 。

Common Resource Fields

在API中,最常见和只读的资源域并不在资源视图中显示。为了让用户更容易浏览资源域列表，我们将这些资源域从列表中移除。

域	类型	注释
accountId	account	相关联账号的唯一识别符。
created[TS]	date	资源创建的起始日期/时间。
id	string	资源的唯一识别符。
kind	string	资源更具体的类型划分。
removeTime	date	资源被移除的时间, 如果资源没有被移除,则该值将为null。
removed	bool	如果资源已经被移除,则为真。已移除的资源不会回到collection列表,但能单独通过ID来检索。
transitioning	enum	资源转换的状态: `yes` , `no` ,或者 `error` 。
transitioningMessage	string	转换过程中出现的信息，或者产生的错误。
transitioningProgress	int	转换期间的进度估算，范围是0-100。
uuid	string	资源的通用唯一识别码。这在Rancher安装过程中始终保持唯一。

资源类型

账号
Rancher上所有资源要么被一个账户(account)拥有，要么由该账户创建。

api密钥
如果启动了访问控制，API密钥(apiKey)将会提供访问Rancher API的能力。由每个环境(environment)创建访问密钥(accesskey)以及秘密密钥(secretkey)对能够直接用来调用API或者和rancher-compose一起使用。

证书
证书(certificate)用于在SSL终端添加到负载均衡器(loadbalancer)。

容器
容器(container)用来表示主机上的一个Docker容器。

dns服务
API中的"dns服务"(dnsService)在用户界面和Rancher文档中被称为服务别名。我们在API文档中使用用户界面术语来称呼。一个服务别名有允许为你可被发现的服务添加DNS记录的能力。

环境
API中的"环境"(environment)在用户界面和Rancher文档中被称为栈(stack)。我们在API文档中使用用户界面术语来称呼。Rancher栈(stack)和docker-compose工程反映了同样的概念。Rancher栈(stack)代表一组构建典型应用和负载程序的服务。

外部服务
外部服务(externalService)允许添加任意IP或者主机作为服务的能力能够以服务的方式被发现。

主机
主机(host)在Rancher中是资源的最小单元，满足以下最低要求的任何虚拟或物理Linux服务器都能表示自己是一台Rancher主机。 任意支持*Docker 1.10.3*的*Linux*发型版本。 必须能够经由http或者https的方式通过预先配置的端口(默认为8080)访问Rancher Server。 * 必须对属于同一个环境中的主机可路由，来为Docker容器使用Rancher的跨主机网络功能。

身份证明
当Rancher启动了访问控制，身份证明(identity)是一个对象(例如 `ldap_group` , `github_user`)在Rancher的表示。身份证明中的 `externalId` 是在认证系统中用来表示对象的唯一识别符。除非身份证明的角色被作为项目成员(projectMember)返回，否则它总为null。

负载均衡服务
Rancher使用HAProxy实现了一个可管理的负载均衡器(loadbalancer)，能够手动地将规模扩大到其他主机上。通过将负载均衡器添加或者"连接"到基本服务，服务均衡器就能用来为单独的容器分化网络和应用流量。被"连接"的基本服务中所有基本的容器将自动被Rancher注册为服务均衡器(loadbalancer)目标。

机器
无论何时Rancher使用 `docker-machine` 在Rancher中创建主机，就会创建机器(machine)条目。在用户界面上添加任意类型的主机。

挂载
挂载(mount)是指在数据卷和容器中目录位置的关系。

项目
用户界面和Rancher文档把API中的"项目(project)"称为环境(environment)。在API文档中,我们使用用户界面上的术语来描述。所有主机和任何Rancher资源(比如，容器,负载均衡器等)在被创建后都属于一个环境.环境的拥有者决定谁能查看和管理这些资源。Rancher当前支持每个用户能管理和邀请其他用户到他所属的环境，也允许为不同的工作量创建多个环境。例如，你可能想创建一个"开发"和一个分离的"产品"环境，这些环境各自带有自己的一系列资源，以及应用程序部署限制的用户访问。

项目成员
用户界面和Rancher文档把API中的"项目成员(projectMember)"称为环境成员(environmentMember)。环境成员是环境中所有成员的列表。环境成员是一个身份证明(identity)。

镜像仓库
镜像仓库(registry)是镜像存储处的地点。镜像仓库可以是DockerHub，Quay.io，或者是定制的私有镜像仓库。

镜像仓库证书
镜像仓库证书(registryCredential)用来授权镜像仓库(registry)。

结构定义
这是结构定义(schema)资源

服务
Rancher对服务(service)采用了标准Docker Compose术语，同时定义一个基本服务为同一个Docker镜像创建的一个或多个容器。在同样的栈中一旦一个服务(消费者)被链接到另一个服务(生产者)，会自动创建一条映射到每一个容器实例的DNS记录，这条记录能够"正在消费"的服务发现。在Rancher中创建服务的其他好处包括":" 服务高可用 - 拥有*Rancher*自动监控容器状态和维护服务想要的规模的能力。 健康监控 - 拥有为容器健康设置基本监控阈值的能力。

存储池
存储池(storagePool)是参与到共享存储的主机名单。

数据卷
数据卷(volume)能够关联到容器或者存储池。 一个容器能够有多个数据卷，容器能被映射到数据卷所在的容器上的挂载连接。 存储池拥有多个存储卷。数据卷只对那些部署到作为存储池一部分的主机上的容器是有效的。当创建一个数据卷，你不能直接将它关联到存储池。你只需要在指定一个驱动，在分配过程中，Rancher将会将其解析给一个存储池。

Rancher OS

快速安装指南

如果你有特别的RancherOS主机要求，请浏览运行RancherOS指南， 本指南的主要讲述使用docker machine来启动RancherOS和介绍RancherOS能够做什么。

If you have a specific RanchersOS machine requirements, please check out our guides on running RancherOS. With the rest of this guide, we'll start up a RancherOS using docker machine and show you some of what RancherOS can do.

使用Docker Machine 启动RancherOS

Launching RancherOS using Docker Machine

前进之前，你需要安装Docker Machine和VirtualBox。 一旦你安装VirtualBox和Docker Machine，它只需要一个命令就可以运行RancherOS。

Before moving forward, you'll need to have Docker Machine and VirtualBox installed. Once you have VirtualBox and Docker Machine installed, it's just one command to get RancherOS running.

```
$ docker-machine create -d virtualbox --virtualbox-boot2docker-url https://releases.ra
ncher.com/os/latest/rancheros.iso <实例机器名>
```

就是这么简单！你已经启动和运行了一个RancherOS实例。 That's it! You're up and running a RancherOS instance.

要登录到该实例，只需使用 `docker-machine` 命令。 To log into the instance, just use the docker-machine command.

```
$ docker-machine ssh <实例机器名>
```

初识RancherOS

A First Look At RancherOS

在RancherOS中运行两个Docker守护进程。首先是所谓的**system-docker**，这就是RancherOS运行的系统服务，如NTPD和系统日志。您可以使用 `system-docker` 命令来控制**system-docker**守护进程。

另一个守护进程是**docker**，它可以通过使用正常的 `docker` 命令来访问。

There are two Docker daemons running in RancherOS. The first is called **system-docker**, which is where RancherOS runs system services like ntpd and syslog. You can use the `system-docker` command to control the **system-docker** daemon.

The other Docker daemon running on the system is **docker**, which can be accessed by using the normal `docker` command.

当你第一次启动RancherOS，还没有容器运行在'docker `守护进程。不过，如果你对运行` system-docker`的命令，你会看到一些RancherOS系统自己维护的服务。

When you first launch RancherOS, there are no containers running in the `docker` daemon. However, if you run the same command against the `system-docker` instance, you'll see a number of system services that are shipped with RancherOS.

> 注：`system-docker`只能由root用户使用，所以每当你需要使用`system-docker`时，交互使用`sudo`命令。
>
> **Note:** `system-docker` can only be used by root, so it is necessary to use the `sudo` command whenever you want to interact with `system-docker`.

```
$ sudo system-docker ps
CONTAINER ID        IMAGE                       COMMAND                 CREATED
       STATUS              PORTS               NAMES
4710a91a0729        rancher/os-docker:v0.4.4    "/usr/sbin/entry.sh /"   6 minutes ago
       Up 6 minutes                            docker
7ef4fad1612c        rancher/os-console:v0.4.4   "/usr/sbin/entry.sh /"   6 minutes ago
       Up 6 minutes                            console
1b436b6b7fdb        rancher/os-network:v0.4.4   "/usr/sbin/entry.sh /"   6 minutes ago
       Up 6 minutes                            network
2ce47a55d1bd        rancher/os-ntp:v0.4.4       "/usr/sbin/entry.sh /"   6 minutes ago
       Up 6 minutes                            ntp
c2237144ec41        rancher/os-udev:v0.4.4      "/usr/sbin/entry.sh /"   6 minutes ago
       Up 6 minutes                            udev
5373e592dc51        rancher/os-acpid:v0.4.4     "/usr/sbin/entry.sh /"   6 minutes ago
       Up 6 minutes                            acpid
c5d8cd81a94c        rancher/os-syslog:v0.4.4    "/usr/sbin/entry.sh /"   6 minutes ago
       Up 6 minutes                            syslog
```

一些容器在启动时运行，和其余时间，如 `console` ，`docker` 等。这些容器始终运行。

Some containers are run at boot time, and others, such as the `console` , `docker` , etc. containers are always running.

开始使用RancherOS

Using RancherOS

部署一个Docker容器

Deploying a Docker Container

让我们尝试使用 `docker` 守护进程部署一个正常的Docker容器。该RancherO S `docker` 守护等同于其他任何Docker环境，使一切正常Docker命令的工作。 Let's try to deploy a normal Docker container on the `docker` daemon. The RancherOS `docker` daemon is identical to any other Docker environment, so all normal Docker commands work.

```
$ docker run -d nginx
```

你可以看到，nginx镜像的容器启动并运行。 You can see that the nginx container is up and running:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             S
TATUS              PORTS               NAMES
e99c2c4b8b30        nginx               "nginx -g 'daemon off"   12 seconds ago      U
p 11 seconds       80/tcp, 443/tcp     drunk_ptolemy
```

部署一个系统服务容器

Deploying A System Service Container

下面是一个简单的Docker容器安装 `Linux-dash` ，它是用于监控Linux服务器最小的低开销网络信息中心。该Dockerfile将是这样的： The following is a simple Docker container to set up Linux-dash, which is a minimal low-overhead web dashboard for monitoring Linux servers. The Dockerfile will be like this:

```
FROM hwestphal/nodebox
MAINTAINER hussein.galal.ahmed.11@gmail.com

RUN opkg-install unzip
RUN curl -k -L -o master.zip https://github.com/afaqurk/linux-dash/archive/master.zip
RUN unzip master.zip
WORKDIR linux-dash-master
RUN npm install

ENTRYPOINT ["node","server"]
```

使用 `hwestphal/ nodebox` 镜像，它采用了busybox的镜像，并安装 `node.js` 和 `npm` 。我们下载的 `Linux-dash` 的源代码，然后运行服务器。 `Linux-dash` 将默认在端口80上运行。 Using the `hwestphal/nodebox` image, which uses a busybox image and installs `node.js` and `npm` . We downloaded the source code of Linux-dash, and then ran the server. Linux-dash will run on port 80 by default.

要使用 `systemd-docker` 运行这个容器使用下面的命令：

To run this container with system-docker use the following command:

```
$ sudo system-docker run -d --net=host --name busydash husseingalal/busydash
```

在这行命令中，我们使用 `--net= host` 告诉 `system-docker` 不使用容器化容器的网络，并使用主机的网络代替。 在运行容器后您可以通过访问 `http://<IP_OF_MACHINE>` 看到监控服务器。

In the commad, we used `--net=host` to tell `system-docker` not to containerize the container's networking, and use the host's networking instead. After running the container, you can see the monitoring server by accessing `http://<IP_OF_MACHINE` .



为了容器能在系统重启时存活，你可以创建 `` `/opt/rancher/bin/start.sh `` 脚本，并添加Docker启动项以保证每次启动时自动启动。 To make the container survive during the reboots, you can create the `/opt/rancher/bin/start.sh` script, and add the docker start line to launch the docker at each startup.

```
$ sudo mkdir -p /opt/rancher/bin
$ echo “sudo system-docker start busydash” | sudo tee -a /opt/rancher/bin/start.sh
$ sudo chmod 755 /opt/rancher/bin/start.sh
```

使用ROS工具包

Using ROS

可与RancherOS使用的另一种有用的命令是 `ros` ,它可用于控制和配置系统。 Another useful command that can be used with RancherOS is `ros` which can be used to control and configure the system.

```
$ ros -v
ros version 0.0.1
```

RancherOS状态由云配置文件控制。 `ros` 用于编辑系统的配置，参见例如系统的DNS配置：RancherOS state is controlled by a cloud config file. `ros` is used to edit the configuration of the system, to see for example the dns configuration of the system:

```
$ sudo ros config get rancher.dns
- 8.8.8.8
- 8.8.4.4
```

当使用本地控制台的Busybox，控制台的任何更改将重启后会丢失，只有改变为 `/home` 或 `/opt` 将是持久的。控制台始终在每次启动时执行**/opt/rancher/bin/start.sh**。 你可以用 `ros` 启用持久控制台和替换本地Busybox的控制台。 为了使启用Ubuntu的控制台，使用以下命令：

When using the native Busybox console, any changes to the console will be lost after reboots, only changes to `/home` or `/opt` will be persistent. The console always executes **/opt/rancher/bin/start.sh** at each startup. You can use `ros` to enable a persistent console and replace the native Busybox console. In order to enable the Ubuntu console, use the following command:

```
$ sudo ros service enable ubuntu-console
# 您必须在重启以切换到Ubuntu的控制台
# You must reboot in order to switch to the ubuntu console
$ sudo reboot
```

结论

Conclusion

RancherOS是一个简单的Linux发行版非常适合运行Docker。通过采用容器化的系统服务和利用Docker进行管理，RancherOS希望能够为运行容器提供了非常可靠，易于管理的操作系统。

RancherOS is a simple Linux distribution ideal for running Docker. By embracing containerization of system services and leveraging Docker for management, RancherOS hopes to provide a very reliable, and easy to manage OS for running containers.

运行 Rancher OS

工作站

Docker Machine

Vagrant

从 ISO 启动

公有云

AWS

GCE

Azure

裸金属机和虚拟机

iPXE

PXE

安装到硬盘

配置

Configuring RancherOS The configuration of RancherOS is derived from two sources.

1. RancherOS ships with a default configuration. The default configuration cannot be changed, but it can be extended or overridden by cloud-config file.
2. Cloud-config extends and overrides RancherOS default config. Cloud-config is obtained on boot from several sources by the cloud-init program running as a system container inside RancherOS. Additionally the cloud-config is read from disk if you wish to make local changes.

As a convenience we provide the ros config command which makes it easy to modify the Cloud-config on disk.

You can view the entire RancherOS configuration in its entirety by typing sudo ros config export --full.

配置存储

网络

用户

SSH Keys

控制台输出

系统服务

设置 Docker TLS

加载内核模块

安装内核模块

DKMS

自定义内核

构建自定义 RancherOS

Docker

定制 Docker

预打包 Docker 镜像

公有云配置参考

升级

ROS 工具

A useful command that can be used with RancherOS is ros which can be used to control and configure the system. ros requires you to be the root user, so with the rancher user, you will need to use sudo.

SUB COMMANDS

| Command | Description | |————–| ——————————————————————————————| |config, c | Configure Settings | |dev, d | dev spec |env, e | Run a command with RancherOS environment | |service, s | Command Line interface for services and compose. | |os | Operating System Upgrade/Downgrade | |tls | Setup TLS configuration | |install | Install RancherOS to Disk | |help, h | Shows a list of commands or help for one command |

RANCHEROS VERSION

If you want to check what version you are on, just use the -v option.

```
$ sudo ros -v ros version v0.4.0
```

HELP

To list available commands, run any ros command with -h or --help. This would work with any subcommand within ros.

```
$ sudo ros -h
NAME:
    ros - Control and configure RancherOS

USAGE:
    ros [global options] command [command options] [arguments...]

VERSION:
    v0.4.0

AUTHOR(S):
    Rancher Labs, Inc.

COMMANDS:
    config, c   configure settings
    dev, d      dev spec
    env, e      env command
    service, s  Coomand line interface for services and compose.
    os          operating system upgrade/downgrade
    tls         setup tls configuration
    install     install RancherOS to disk
    help, h     Shows a list of commands or help for one command

GLOBAL OPTIONS:
    --help, -h                  show help
    --generate-bash-completion
    --version, -v               print the version
```

ROS 简介

Config

TLS

OS

Service

ENV

Install

Amazon ECS

底层技术

目录加载

为 RancherOS 做贡献

技巧

常见问题

Rancher Server

Rancher Agent/Host

排错